ioProgrammo
PER ESPERTI E PRINCIPIANTI

ANTEPRIMA: VISUAL BASIC 9.0

TUTTE LE NOVITÀ DEL LINGUAGGIO CHE VERRÀ. FATTI TROVARE PREPARATO A MIGRARE...

Rivista + "Le grandi guide di ioProgrammo" n°8 a € 12,90 in più
VERSIONE PLUS RIVISTA+LIBRO+CD €9,90
VERSIONE STANDARD RIVISTA+CD €6,90

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • DICEMBRE 2006 • ANNO X, N.12 (109)

FATE IL VOSTRO VIDEOGIOCO

CON IL NUOVO XNA GAME STUDIO DI MICROSOFT

programmare il tuo sparatutto personale diventa veramente esageratamente semplice!

STRUMENTI
Installiamo il software necessario e integriamolo con Visual Studio

TEORIA
Struttura di un'applicazione. Ecco le classi fondamentali ed il loro funzionamento

TUTTO IL CODICE nel CD

PRATICA
La guida passo passo per realizzare il primo game 2D. Le basi per iniziare a muoversi nella terza dimensione

SOFTWARE "SICURO"? FACCIAMOLO CON C++

10 "linee guida" per gestire bene la memoria ed evitare bug che possono dare origini a Exploit e Virus

C#
CREA UN GENIO PER GLI UTENTI
Impara a programmare i Wizard: le procedure guidate che aiutano ad usare il software

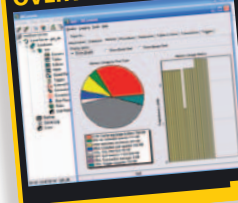
VB.NET
ESPRESSIONI REGOLARI
Ecco il trovatutto che scova le stringhe nascoste nel testo in un battibaleno

ASP.NET
JAVASCRIPT E .NET
Non sono la coppia più bella del mondo, ma noi li mettiamo d'accordo

ECOMMERCE CON PAYPAL
Sviluppa un intero sistema di commercio elettronico spendendo poco o niente

JAVA
IL CODICE DEL GIOCO IN BORSA
Prendi le informazioni dalla rete, mettile in un DB e il gioco è fatto

VISUAL BASIC
PERSONALIZZA IL TUO AMBIENTE
Migliora l'IDE e automatizzalo con le funzioni che servono a te

ESTENDI SQL SERVER
Usa Visual Basic 6 con il tuo database preferito e inoltre rendilo più potente

ASP.NET
GESTIAMO LE MAILING LIST
Dall'iscrizione all'invio della posta. Creiamo una web application che fa tutto

REPORT NO PROBLEM
Svincoliamoci dai programmi commerciali. Ecco il codice che azzera le spese

JAVA
AJAX? WEB 2.0 SÌ FATICA NO
Arriva Echo2 il framework che ti porta nella nuova rete senza farti impazzire

SOLUZIONI: PROGRAMMA CON GLI ALGORITMI GENETICI UN ROBOT INTELLIGENTE CHE SVILUPPA AL POSTO TUO

Anno X - N.ro 12 (109) - Dicembre 2006 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, L. Buono, A. Galeazzi, F. Grimaldi, F. Smelzo, A. Pelleriti, M. Locuratolo, L. Corias

Segreteria di Redazione: Veronica Longo

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 20% sul prezzo di copertina di €75,90 · ioProgrammo con Libro (11 numeri) €75,90 sconto 30% sul prezzo di copertina di €108,90 Offerte valide fino al 31/12/06 costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Servizio Abbonati:
☎ tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Novembre 2006

UN PASSO OLTRE

La domanda è: quanto ci rimane ancora da inventare? In programmazione la risposta è: "praticamente tutto". Ma è anche vero che il livello di qualità raggiunto nel campo dello sviluppo dei videogame è davvero elevato. Stesso discorso vale per la sicurezza, per i gli antivirus, per i gestionali, per i cms, per l'analisi matematica. Ad oggi, nel 2006, possiamo dire che il contributo che noi programmatori diamo ogni giorno al miglioramento della qualità del software è davvero impressionante. Se è vero che l'informatica, i computer, e il software in generale sono quegli elementi che forniscono la maggiore propulsione allo sviluppo, possiamo anche affermare che proprio noi sviluppatori rappresentiamo l'elemento cardine attorno al quale si giocano le sorti di settori economici, scientifici, matematici e molto altro ancora. In tutti quei campi dove si usa un computer è molto probabile che uno di noi abbia fornito il proprio essenziale contributo. E' ora di prendere coscienza del nostro ruolo e forse di far valere un po' di più il nostro lavoro. Ma non è un discorso economico-politico quello in cui mi voglio lanciare, quanto piuttosto tracciare una linea immaginaria lungo la quale noi sviluppatori potremmo iniziare a muoverci con più scioltezza. Quali sono i nostri prossimi traguardi? L'idea è che ormai disponiamo di una serie di strumenti veramente efficaci, fatti da altri sviluppatori per noi. I compilatori, gli IDE, i database, gli strumenti di progettazione che abbiamo fra le mani ci consentono di non mettere limiti alla nostra creatività. E allora quali sono le nuove mete verso cui spingerci? La risposta è incerta, e potrebbe tradursi in un ipotetico "nella risoluzione di ogni problema che un cliente possa porci". Tuttavia, in linea generale, esiste un settore che a mio avviso potrebbe essere più interessante degli altri, ed è quello dell'intelligenza artificiale. Non ancora intesa come emulazione del comportamento umano, quanto come capacità per un automa di autoapprendere e reagire con soluzioni matematiche a soluzioni non prevedibili. Probabilmente il nostro prossimo traguardo sarà proprio quello relativo alla creazione di modelli decisionali perfetti. Tremenda come prospettiva, ma anche molto affascinante…

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre \soft\codice\ e \soft\tools\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

ed ora... fate il vostro VIDEOGIOCO

CON IL NUOVO XNA GAME STUDIO DI MICROSOFT

Programmare il tuo sparatutto personale diventa veramente esageratamente semplice!

- ✔ Strumenti: Intalliamo il software necessario, e integriamo con Visual Studio
- ✔ Teoria: Struttura di un'applicazione. Ecco le classi fondamentali. ed il loro funzionamento
- ✔ Pratica: La guida passo passo per realizzare il primo game 2D

pag. 14

SOFTWARE "SICURO FACCIAMOLO CON C++

10 "linee guida" per gestire bene la memoria ed evitare bug che possono dare origini a Exploit e Virus

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

http://forum.ioprogrammo.it

RIVISTA + CD-ROM in edicola

TURBO DELPHI FOR .NET

Un ritorno storico

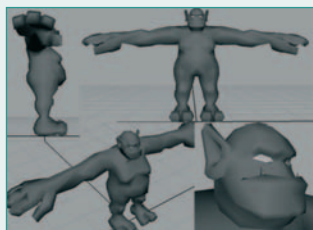
Dieci anni fa Delphi ha segnato il punto di svolta nelle tecnologie di sviluppo. Nessuna innovazione ha segnato tanto quanto l'introduzione degli IDE RAD una rivoluzione nel campo dello sviluppo. E fu Borland con il suo Delphi a portare questa innovazione. Dopo anni in cui Borland si è quasi totalmente dedicata al segmento "business" ecco che torna con una suite di ambienti dedicati alla produttività individuale fra cui spiccano "Turbo C#" di cui ci siamo occupati nel numero scorso e "Turbo Delphi" di cui ci occupiamo in questo numero. Si tratta di prodotti eccezionali che meritano di essere installati sulle macchine di tutti gli sviluppatori. Nella versione turbo sono completamente completi e gratuiti e godono di caratteristiche di eccellenza che meritano di essere provate. Necessita registrazione sul web per ottenere la chiave di sblocco che lo abilita all'uso gratuito e completo per un tempo illimitato.

Prodotti del mese

Animadead 2.0

L'animatore di scheletri

Diciamo che volete realizzare un'animazione 3D completa. Da dove partire? AnimaDead utilizza un approccio originale. Quello che dovete fare è disegnare uno scheletro del personaggio da animare, fatto di barrette e giunture. A questo punto programmate il movimento dei singoli pezzi tenendo conto di come sono congiunti fra loro. Infine ricoprite il tutto con i tessuti. L'idea è ben realizzata e l'approccio è veramente molto interessante. Su ioProgrammo ce ne siamo occupati già in passato con un articolo approfondito, la tecnica rimane decisamente affascinante. Nonostante qualche piccola lacuna nella libreria, il prodotto rimane ben fatto. Inoltre la disponibilità dei sorgenti fa si che eventuali esigenze personali possano essere risolte modificando il codice originale

[pag.105]

Irrlicht 1.1

Il motore 3D per eccellenza

In questo nuovo numero di ioProgrammo presentiamo XNA, il nuovo game engine di Microsoft, ma quali sono i suoi rivali? Sicuramente irrlicht è uno di questi! E' vero per usare irrlicht bisogna avere qualche nozione di C++ ma l'engine è talmente ben congegnato che la difficoltà tecnica è veramente bassa. Su ioProgrammo ci siamo occupati a lungo di Irllicht in passato e vi abbiamo persino dedicato un libro "Programmare VideoGiochi" di Alfredo Marroccelli. Irrlicht rimane uno dei principali strumenti per sviluppare giochi 3D realistici ed estremamente veloci. E' vero che è necessario avere qualche conoscenza di C++, tuttavia una volta presa dimestichezza con l'SDK tutto procede con estrema velocità. Lo schema base è veramente semplice

[pag.105]

Ruby 1.8.5

Il vero nuovo che avanza

In America è il linguaggio che maggiormente ha scalato i vertici delle classifiche di utilizzo nell'ultimo anno. In Italia sta giungendo rapidamente come un ciclone a fare capolino nel panorama dello sviluppo. Bello, elegante, con una curva di apprendimento praticamente nulla, si tratta di un linguaggio di scripting che ottimamente si presta ad essere utilizzato sia sul Web sia in modo standalone. Recentemente ce ne siamo occupati in un bell'articolo di Paolo Perrotta, che mostrava come utilizzarlo per programmare un plugin per Skype messenger. Il linguaggio è elegante e raffinato, fa un uso intensivo delle liste, ma anche della programmazione ad oggetti. Si presta bene allo sviluppo di qualunque tipo di applicazione e recentemente proprio un framework basato su Ruby: RubyOnRails ha vinto il premio come miglior framework per lo sviluppo web

[pag.106]

Python 2.5

L'ex giovane rampante

Python è stato considerato per lungo tempo il nuovo che avanza. Attualmente non lo si può più definire in questo modo, Python è ormai un linguaggio stabile e completo che trova applicazione in un gran numero di progetti. Se ne parla sempre di più in campo industriale come su Internet. Soprattutto un gran numero di applicazioni anche in ambiente Windows girano ormai grazie a Python e presentano interfacce grafiche ottimamente strutturate. Ciò nonostante Python rimane un grande linguaggio di scripting adatto a gestire in modo completamente automatico buona parte di un sistema operativo sia esso Linux o Windows. Attualmente il linguaggio è in forte espansione tanto che viene utilizzato per la gestione del sistema operativo sia in ambiente Windows che in Ambiente Linux

[pag.106]

I contenuti del libro

Lavorare con ADO.NET

Ci sono le applicazioni, ma la maggior parte servirebbero a poco se non fosse possibile immagazinare i dati in una qualche maniera. Esistono i database, ma sono tanti, ed ognuno gode delle proprie peculiarità. Impossibile conoscerli tutti. E infine esistono gli utilizzatori che scelgono un database piuttosto che un altro sulla base delle proprie esigenze. Come scrivere software in grado di utilizzare qualunque database? La risposta è semplice e si chiama ADO.NET. Si tratta della tecnologia indispensabile "middleware" fra i database e il linguaggio. Conoscere ADO.NET significa avere in mano la chiave per sviluppare applicazioni che accedano in maniera ottimale ad una base di dati. Solo in questo modo si ottengono performance e funzionalità. Michele Locuratolo è un'ottima guida che gradualmente ci insegna come sfruttare al meglio ADO.NET

IMPARARE SUBITO A PROGRAMMARE CON LA TECNOLOGIA CHE GESTISCE I DATABASE IN AMBIENTE WINDOWS

- **La teoria delle basi di dati**
- **Connettersi ed usare un database**
- **I componenti per la visualizzazione**
- **Lo scambio dei dati e l'integrazione con XML**

News

INTERNET EXPLORER 7 GIÀ A RISCHIO

È appena arrivato sul mercato con tutto il suo carico di novità e i pirati dell'informatica ne hanno messo a nudo le prime debolezze. Il record di velocità nella segnalazione di una falla di sicurezza in IE7 lo guadagna Secunia che nel suo bollettino del mese di Ottobre ha messo in guardia gli utenti contro una vulnerabilità legato al Phishing. La vulnerabilità riguarderebbe la possibilità di visualizzare una finestra popup particolarmente insidiosa il cui URL di provenienza potrebbe essere parzialmente mascherato. Si tratta di una vulnerabilità non particolarmente feroce a dire il vero, ma che lascia stupefatti per la velocità con cui è stata rilevata.

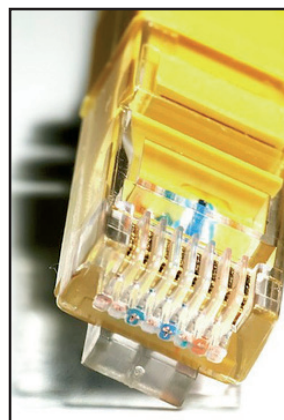
ADO SOTTO INCHIESTA

Il famoso componente che consente la connessione a database è in questi giorni al centro dell'attenzione. La causa scatenante di tante polemiche sul componente più famoso della programmazione è dovuta ad una segnalazione di US-CERT secondo la quale, inviando ad ADO un certo numero abbastanza elevato di false query SQL si potrebbe arrivare ad eseguire del codice remoto. Attualmente il baco è in fase di investigazione da parte di Microsoft che tuttavia non ha ancora né confermato né smentito. Ovviamente, vista l'importanza che riveste questo elemento, la preoccupazione fra i programmatori è elevata. Il 90% delle applicazioni Web si basa proprio sulla connessione ai database.

PHP BATTE TUTTI

A dirlo è una ricerca condotta dalla statunitense Ohlo, compagnia che dal 2004 si occupa della sensibilizzazione nella promozione del software Open Source. La Ohlo ha recentemente pubblicato un'interessante statistiche che prende in considerazione 4 dei linguaggi più usati nello sviluppo di codice aperto: PHP, Ruby, Python e Perl. Prima di ogni cose Ohlo ha considerato il numero nuove righe di codice elaborate negli ultimi anni. PHP sarebbe di gran lunga il linguaggio che maggiormente si è sviluppato in questo senso. La conclusione appare ovvia se si pensa al proliferare di web application e fork che ogni giorno popolano Internet e sono sviluppate appunto in PHP. La seconda analisi fa riferimento al numero di nuovi programmatori per linguaggio, ed anche in questo caso è sempre PHP ad avere la meglio. La sorpresa arriva però dal numero di nuovi progetti, qui è Ruby ad avere la meglio con un +15% particolarmente significativo rispetto al 5% mediamente detenuto dagli altri linguaggi. In conclusione mentre PHP sembra sempre più consolidare progetti nati qualche anno fa che hanno assunto ormai dimensioni generose, il linguaggio emergente sembra essere Ruby. In italia la situazione è ancora leggermente diversa, di fatto ancora Ruby non è arrivato in modo prepotente, ma considerato il ritardo con cui ci muoviamo rispetto all'America non dubitiamo che da qui a qualche tempo vedremo questo nuovo linguaggio fare capolino in molte delle nostre applicazioni. A fare la fortuna di questo linguaggio, un'eleganza innata e una curva di apprendimento molto bassa. Il resto lo fanno le librerie e i costrutti piuttosto avanzati che rendono molto semplice la vita a chi programma.

LA BANDA LARGA È SEMPRE PIÙ TELECOM

La preoccupazione è forte da pare di AIIP, ovvero l'associazione italiana degli Internet Provider. Il pericolo di monopolio è evidente e tangibile. Così si è tornati a chiedere al garante delle telecomunicazioni una maggiore presenza nella verifica delle condizioni che garantiscano un vero regime di concorrenza. La paura da parte dei provider è sempre la stessa. Telecome è al contempo grossista della banda larga e anche venditore al dettaglio. L'attuale legge fa si che esista un calcolo preciso nella determinazione dei prezzi all'ingrosso, ma tale calcolo fa si che il prezzo al consumatore debba per forza di cose essere livellato per tutti i venditori di banda italiana e questo fa si che nessuno realmente riesca a determinare dei prezzi che siano competitivi sul mercato. La situazione è ben spiegata da Stefano Quintarelli, presidente dell'AIIP nel suo blog http://quinta.typepad.com/blog/2006/10/aiip_associazio.html. Quintarelli evidenzia come la Telecom detenga una quota di mercato al dettaglio eccessivamente elevata e come i prezzi siano significativamente più alti. L'appello si conclude con una richiesta di intervento da parte di AGCOM.

ADDIO BLUETOOTH ARRIVA WIBREE!

A sviluppare questo nuovo protocollo di comunicazione, sarà ovviamente il colosso norvegese Nokia. L'annuncio è di quelli che scotta. Le periferiche BluetTooth attualmente disponibili sono veramente tantissime e non si conoscono ancora i dettagli con cui Nokia intende sviluppare questa nuova tecnologia. Le prime indiscrezioni parlano della volontà di produrre una tecnologia in grado di lavorare con basse potenze in modo tale da favorire un risparmio del consumo energetico. L'obiettivo è quello di allargare il bacino di utenza della piattaforma wireless. Oltre gli attuali palmari, telefonini, tastiere mouse, modem etc. il nuovo Wibree potrebbe essere utilizzato anche su oggetti di dimensioni decisamente piccole e con un'alimentazione tale che allo stato attuale non consente l'utilizzo del bluetooth. Si pensi per esempio agli orologi. L'idea è quella ancora una volta di mettere in comunicazione fra loro un gran numero di periferiche. La velocità di trasferimento, sempre secondo indiscrezioni potrebbe aggirarsi intorno a 1MBPS. Infine per quanto riguarda la progettazione i ben informati parlando di un circuito che almeno inizialmente potrebbe prevedere la doppia modalità Bluetooth/Wibree. Le ipotesi sui probabili scenari che l'avvento di una tecnologia del genere potrebbe aprire sono attualmente pura fantascienza. Al solito occorrerà molto tempo per comprendere le effettive dimensioni del progetto. Nel frattempo diverse aziende sono corse ad aiutare Nokia, citiamo per tutte la sola Epson. Ed esiste già un concorrente: Zigbee.

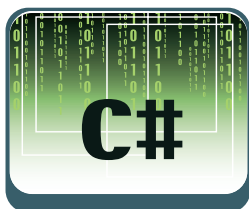
AL VIA ASP.NET AJAX CORE V1.0

Già da tempo Microsoft stava lavorando ad un prodotto che rendesse semplice la programmazione in tecnologia Ajax per l'ambiente .NET. Il nome in codice di questo progetto era Atlas. Recentemente proprio come evoluzione di Ajax è stata rilasciata la prima versione Beta dell'Asp.NET Ajax core V1.0, che rappresenta la via di Microsoft verso il web 2.0. Le novità riguardano un migliorato supporto verso la compatibilità con tutti i browser, da Safari a Firefox. Inoltre sono stati inseriti ben 28 controlli che nelle intenzioni di Microsoft dovrebbero rendere la programmazione Ajax del tutto trasparente da parte del programmatore. In buona sostanza utilizzando questo framework si è certi di sviluppare applicazioni Web 2.0 pur non conoscendo i dettagli della tecnologia. Se si pensa che buona parte del futuro di Internet si gioca sul successo di questa tecnologia appare evidente la voglia di Microsoft di investire in questo settore.

TEMPO DI CRISI PER MYSPACE

Nell'era del Social Networking, i leader portano i nomi di YouTube e MySpace. Ma si sà che ai nostri giorni tutto è più rapido. Anche il tempo che intercorre fra l'ascesa e il declino può subire incredibili accelerazioni. Così mentre YouTube passa in un battibaleno dall'anonimato alla leadership, tanto da essere acquistato dal gigante Google, allo stesso modo MySpace fa conoscenza con l'atmosfera della crisi. Nulla di irrimediabile ancora. Tuttavia il gigante di Microsoft accusa i colpi che gli vengono continuamente sferrati dai pirati dell'informatica, sotto la forma specialmente del Phishing. Gli utenti delusi, decidono di cambiare aria. MySpace accenna una timida reazione ma non sarà facile rimettere in sesto il disastrato spazio pubblico. Tuttavia Microsoft non è certo un novellino e non si lascerà mettere nel sacco dai pirati dell'informatica.

XNA E GAME STUDIO EXPRESS

L'INDUSTRIA DELL'INTRATTENIMENTO HA TEMPI SEMPRE PIÙ STRETTI, I VIDEOGIOCATORI ESIGENZE SEMPRE PIÙ GRANDI. MICROSOFT VIENE INCONTRO A PROFESSIONISTI E APPASSIONATI CON UN FRAMEWORK CHE AUMENTA PRODUTTIVITÀ E DIVERTIMENTO

Con una mossa a sorpresa, auspicata da pochi addetti ai lavori ma ai più giunta come un fulmine a ciel sereno, Microsoft ha annunciato durante la conferenza Gamefest 2006 la disponibilità della beta di XNA GSE (Games Studio Express). XNA GSE è una versione per hobbisti e sviluppatori indipendenti del tool XNA che permetterà lo sviluppo multi-piattaforma di applicazioni videoludiche su Windows ed Xbox 360, mentre un possibile port su piattaforme mobile (PDAs e smartphones) non è da escludere secondo le recenti dichiarazioni rilasciate da Microsoft (http://msdn.microsoft.com/ directx/xna/faq). XNA Games Studio Express integra tutti gli strumenti forniti da XNA in Visual C# Express 2005, mettendo a disposizione specifici templates e designers utili per la programmazioni di applicazioni videoludiche. Nonostante XNA fosse nell'aria da tempo (la prima presentazione su XNA ad opera di J. Allard risale a GDC2004), diverse "sorprese" hanno positivamente impressionato gli utenti evoluti e gli appassionati:

- Oltre ad una versione Professional indirizzata ai vari Game Studios che vedrà la luce nella primavera 2007, è stata rilasciata la sopraccitata versione Express che rappresenta in effetti, nonostante le limitazioni che andremo ad analizzare tra breve, uno SDK per lo sviluppo multi-piattaforma di videogames su Windows ed Xbox 360.

- La scelta di C#, nonostante sia stata criticata dalla parte più conservatrice ed "hardcore" del panorama dello sviluppo videoludico ancora legata prevalentemente a C++ unamanged, è molto coraggiosa. Ciò che Microsoft conta di realizzare è un nuovo passaggio epocale nel mondo dello sviluppo videoludico che si succederebbe alla transizione da Assembler a C in primis, ed a quella da C a C++ in secundis.

- XNA includerà sia delle funzionalità dedite allo sviluppo di codice riusabile per videogames che una "content pipeline", ossia un insieme di strumenti volti a gestire i contenuti del videogioco stesso ed il progetto in generale. L'enfasi posta da Microsoft sulla necessità di avere strumenti evoluti per la gestione di un progetto videoludico nel suo insieme è motivata dalla preoccupante tendenza al rialzo evidenziata dai budget di sviluppo per videogames di ultima ed (ancora più) di prossima generazione.

Al di là delle dichiarazioni di Microsoft, facenti perno su di una campagna di marketing che promette di "democratizzare lo sviluppo videoludico", senza ombra di dubbio è la prima volta che sviluppatori indipendenti, università, e perché no persino appassionati, possono mettere le mani su un vero proprio kit di sviluppo per console.
Sony aveva tentato anni or sono, col progetto Net Yaroze, di permettere lo sviluppo di videogames per la Playstation ma, complice il costo che si aggirava attorno ai 750$ US, l'iniziativa non decollò.
Oltretutto, mentre Net Yaroze era in effetti una versione speciale della Playstation dotata di tool di sviluppo e debugging, XNA è semplicemente del software di sviluppo il cui risultato può essere testato su di una qualunque comune Xbox 360.
Naturalmente la versione Express di Game Studio presenta delle limitazioni che invece non saranno presenti nella versione Professional: anche se la commercializzazione di videogames prodotti con XNA GSE è permessa su Windows, ciò non è possibile per la 360.
È invece possibile, mediante l'iscrizione ad un Creator's Club alla relativamente modica cifra di 99$ US per anno, scambiarsi le rispettive creazioni, siano esse codice, modelli 3D o quant'altro.
Da non sottovalutare infine l'enorme vantaggio apportato ad Università che potranno inserire corsi basati su XNA nel loro curriculum ed, in ultima analisi, permettere agli studenti di effettuare il deployment delle loro creazioni sulla console senza per questo dover ricorrere a strumenti costosi o arcani.
Attualmente le Università non sono in grado di procurarsi strumenti di sviluppo per console, da un lato per via dei costi e dall'altro per via dell'impos-

Conoscenze richieste
Conoscenze base di C#, Conoscenze di base di Object oriented programming

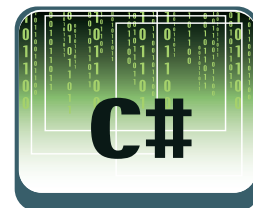
Software
WindowsXpSP2, Visual C# Express, Game Studio Express Beta1

Impegno

Tempo di realizzazione

sibilità di soddisfare tutti i criteri richiesti per fregiarsi del titolo di sviluppatore accreditato.
Questo articolo getterà luce sui pilastri fondamentali su cui XNA poggia: nel prossimo paragrafo dettaglieremo il processo di configurazione attualmente in uso per XNA e i prerequisiti software e hardware per l'installazione ed il suo corretto utlizzo.
Successivamente descriveremo la struttura tipica di una applicazione XNA per poi giungere prima ad un esempio 2D basato su sprites, e poi ad un semplice esempio 3D. Chiuderemo la nostra disamina con delle note sulle correnti limitazioni presenti nella beta di XNA e dei commenti su quello che possiamo attendere per il futuro di questa ed altre tecnologie ad essa collegate.

SETUP DI XNA

XNA è una tecnologia che, sebbene sia rivoluzionaria per quanto riguarda i framework di sviluppo di applicazioni d'intrattenimento, ha basi solide: il .NET Framework 2.0 e DirectX 9.0c e Visual C# Express come ambiente di sviluppo.
Al fine, dunque, di portare avanti il setup di Game Studio Express, si dovrà preparare il proprio PC con gli oppurtuni software; il primo è senz'altro il runtime di .NET 2.0 che scaricheremo dal seguente URL http://www.microsoft.com/downloads/details.aspx?familyid=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=en .
Terminato il setup di questo, sarà necessario installare ed attivare preventivamente anche Visual C# Express, seguendo le indicazioni di default tranne per l'installazione di SQLServer Express ed MSDN Library Express Edition.
Puntando il browser su http://msdn.microsoft.com/vstudio/express/visualcsharp/download/ scaricheremo un setup che analizzerà il pc e scaricherà le componenti necessarie al setup completo.
Al termine, sarà necessario seguire i link proposti per la registrazione della copia di Visual C# Express sul sito Microsoft; oltre ai vantaggi di eseguire la registrazione (librerie di icone e immagini gratuite), questa è obbligatoria dopo 30 giorni ed inoltre Game Studio Express potrebbe avere dei malfunzionamenti se Visual C# Express non fosse già registrato nel momento in cui viene installato.
Le DirectX 9.0c dovreste averle già installate sul vostro PC se avete un sistema operativo Microsoft WindowsXP con SP2 e costantemente aggiornata.
In caso contrario, l'URL per il solo runtime è il seguente: http://www.microsoft.com/downloads/details.aspx?FamilyId=0A9B6820-BFBB-4799-9908-D418CDEAC197&displaylang=en
Il nostro pc è quasi pronto allo sviluppo di applicazioni XNA: si scaricherà direttamente il programma di installazione di Game Studio Express attraverso questo link: http://www.microsoft.com/downloads/details.aspx?FamilyId=21E979E3-B8AE-4EA6-8E65-393EA7684D6C&displaylang=en .
Ricordiamo che quest'ultimo indirizzo, così come l'articolo, fa riferimento alla versione Beta1 dell'ambiente di sviluppo XNA.
A questo punto non ci rimane che lanciare Game Studio Express ed iniziare la nostra avventura in compagnia di Game Studio Express e C#.

STRUTTURA DI UNA APPLICAZIONE

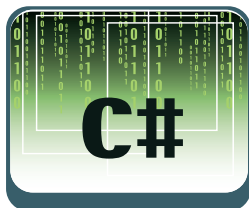
Una tipica applicazione XNA creata in Games Studio Express eredita dalla classe *Microsoft.Xna.Framework.Game* ed ha la responsabilità di effettuare l'ovverride di specifici metodi e delegati elencati oltre. Va notato che fortunatamente Games Studio Express mette a disposizione un template per un progetto di tipo WindowsGame (con la versione per 360 a seguire nella versione finale di Dicembre 2006) che genererà gran parte dello scheletro dell'applicazione, lasciandoci il compito di implementare suddetti metodi, ritoccando dettagli minimi ed inserendo metodi mancanti a seconda delle nostre esigenze. I passi necessari a creare una applicazione XNA minimale sono i seguenti:

1. Creare un progetto di tipo WindowsGame selezionando File/New Project/Windows Game (XNA). Del codice verrà automaticamente generato per una classe che eredita da *Microsoft.Xna.Framework.Game*.

2. Effettuare l'override dei metodi Draw, *Update* e *OnStarting*: Il primo metodo effettua il rendering di un nuovo frame, il secondo aggiorna lo stato della logica di gioco, mentre l'ultimo si occupa di effettuare il setup del gioco prima che venga effettivamente lanciato, generalmente caricando risorse da file, ecc.

3. Nonostante anche questo passo venga automatizzato dal template presente in GSE, è bene sapere che una variabile membro di tipo GraphicsComponent deve essere creata ed inizializzata, per poi essere aggiunta alla lista dei *GameComponents* usati dell'applicazione. La classe *GraphicsComponent* si occupa di astrarre tutti i dettagli relativi alla gestione del device grafico e consentira' a regime di gestire un display sia su Windows che su Xbox 360.

4. Creare un nuovo *EventHandler* che si occupi di gestire l'evento *DeviceReset*: tutto il codice che si occupa di creare ed inizializzare risorse dipendenti da un *GraphicsComponent* dovrebbe esse-

re inserito nel gestore di eventi, per evitare che il reset del device sollevi eccezioni a runtime dovuti a risorse mancanti dopo il reset.

5. Creare un'istanza della classe derivata da *Microsoft.Xna.Framework.Game*.

6. Infine, chiamare il metodo *Run* sulla suddetta istanza.

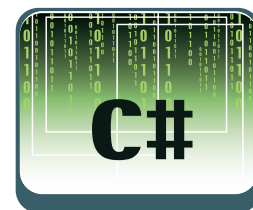
Il template WindowsGame genererà una classe che corrisponde al vero e proprio *entry point* dell'applicazione, all'interno del cui metodo *Main* viene creata l'istanza del nostro gioco e mandata in esecuzione come descritto nei passi 5 e 6.
Da notare inoltre che attualmente il template presente nella Beta di XNA GSE genera una classe partial che eredita da *Microsoft.Xna.Framework.Game* in due file, il primo contenente il codice d'inizializzazione del *GraphicsComponent*, ed il secondo tutto il resto del codice. Il motivo è dettato dalla presenza di un designer all'interno del quale possono essere inseriti le componenti presenti nel sistema. Il concetto di componente non è sicuramente ignaro ai più che avranno manipolato disparati tipi di controlli (presenti nel toolbox di Visual Studio) come le Windows Forms ad esempio.
In termini generali, cari ai cultori dell'Ingegneria del Software, potremmo definire una componente come una unità di codice auto-contenuta ed in grado di svolgere una serie di funzionalità (o servizi) messi a disposizione dell'applicazione che la ospita, mediante una opportuna API.
Questa pratica aumenta il riuso del codice, diminuendo al contempo il numero di bug introdotti nel codice, in maniera tale da consentire un aumento in termini di produttività e modularità del codice prodotto. In tal senso i *GameComponents* non fanno eccezione alla regola: mettendo a disposizione altrettanti metodi *Update* e *Draw*, presente anche in una classe di tipo Game, è possibile modularizzare un gioco e riusare componenti generiche anche in giochi diversi. Una breve introduzione ai GameComponents è contenuta all'interno della documentazione di XNA, mentre un ottimo tutorial e dettagliato e' presente sul blog del team di XNA (http://blogs.msdn.com/xna/archive/2006/08/31/734204.aspx).

APPLICAZIONE DI ESEMPIO 2D

Il primo passo verso lo sviluppo di un'applicazione 2D, passa attraverso la definizione di un oggetto che, molti di voi, avranno visto gironzolare per lo schermo durante le svariate ore passate davanti ad un videogame: lo Sprite.
Uno Sprite è un'immagine bidimensionale (fissa o animata) all'interno di una scena.
XNA ci mette a disposizione alcuni strumenti di supporto alla gestione di questo tipo di oggetti; uno di questi è *SpriteBatch* . Questo oggetto ci permette di gestire uno o più sprite che condivideranno la stessa scena. L'esempio che andremo a sviluppare in questo paragrafo ci permetterà di spostare uno sprite a zonzo per lo schermo utilizzando solo poche istruzioni.
Iniziamo con il creare un nuovo progetto di tipo *WindowsGame* dal menu *File --> New --> Project* per poi scegliere proprio *WindowsGame*.
Il primo passo, quindi, sarà l'introduzione di variabili membro all'interno della classe *Game1* (automaticamente creata da Game Studio Express):

```
partial class Game1 : Microsoft.Xna.Framework.Game
{
    Texture2D myTexture;
    int spriteX = 0;
    int spriteY = 0;
    int m_dSpriteHorizSpeed = 1;
    int m_dSpriteVertSpeed = 1;
    SpriteBatch spriteBatch;
    [...]
```

con queste semplici istruzioni abbiamo dichiarato le variabili che ci permetteranno di caricare l'immagine che rappresenterà il nostro sprite (*myTexture* di tipo *Texture2d*), la sua posizione, inizialmente in alto a sinistra (*spriteX* e *spriteY*), la sua velocità di spostamento orizzontale e verticale (*m_dSpriteHorizSpeed* e *m_dSpriteVertSpeed*) ed, infine, l'oggetto *SpriteBatch* che disegnerà lo sprite vero e proprio sullo schermo. Continuando, possiamo fare l'override del metodo *OnStarting* della classe *Game* in modo da poterci sottoscrivere all'evento *DeviceReset*; l'eventhandler ci permette di dare un "volto" al nostro Sprite, caricando una immagine (in formato jpg, png o bmp) attraverso il metodo statico *FromFile* della classe *Texture2D*.
Ricordiamo che *OnStarting* viene chiamato all'inizio del ciclo di vita del gioco e quindi una sola volta; sottoscrivendo l'evento *DeviceReset* ci assicuriamo che lo sprite abbia sempre il suo "volto" aggiornato.

```
protected override void OnStarting()
{
  base.OnStarting();
  graphics.GraphicsDevice.DeviceReset += new
            EventHandler(GraphicsDevice_DeviceReset);
  LoadResources();
}
void LoadResources()
{
```

```
    myTexture =
            Texture2D.FromFile(graphics.GraphicsDevice,
                                    "home_32.png");
    spriteBatch = new
            SpriteBatch(graphics.GraphicsDevice);
}
```

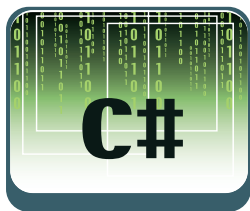
Il passo successivo è l'aggiornamento della posizione dello sprite, attraverso il cambio della sua posizione: lo facciamo nel metodo indicato dal framework come il più indicato per il cambiamento delle logiche di gioco e e degli oggetti che popolano la scena ovvero *Update*.
Nel momento in cui lo sprite dovesse arrivare sui bordi della scena stessa, ritornerà indietro proprio come la pallina del tanto caro Pong.

```
protected override void Update()
{
    float elapsed = (float)ElapsedTime.TotalSeconds;
    // TODO: Add your game logic here
    UpdateSprite();
    UpdateComponents();
}

void UpdateSprite()
{
    //move the sprite by speed
    spriteX += m_dSpriteHorizSpeed;
    spriteY += m_dSpriteVertSpeed;
    int MaxX = Window.ClientWidth - myTexture.Width;
    int MinX = 0;
    int MaxY = Window.ClientHeight -
                            myTexture.Height;
    int MinY = 0;
    //check for bounce
    if (spriteX > MaxX)
    {
        m_dSpriteHorizSpeed *= -1;
        spriteX = MaxX;
    }
    else if (spriteX < MinX)
    {
        m_dSpriteHorizSpeed *= -1;
        spriteX = MinX;
    }
    if (spriteY > MaxY)
    {
        m_dSpriteVertSpeed *= -1;
        spriteY = MaxY;
    }
    else if (spriteY < MinY)
    {
        m_dSpriteVertSpeed *= -1;
        spriteY = MinY;
    }

}
```

Finalmente, disegniamo tutto a schermo: il metodo di cui fare l'override è proprio il Draw: qui XNA si preoccuperà di preparare la scena da mostrare all'ansioso videogiocatore.
Proprio per non farlo attendere e non deluderlo, questo metodo dovrà essere il più efficiente possibile. Daltronde, parliamo di software che, fino a pochi anni or sono, venivano scritti con parti in assembler e con interazioni "intime" con l'hardware sottostante.

```
protected override void Draw() {
    // Make sure we have a valid device
    if (!graphics.EnsureDevice())
        return;
graphics.GraphicsDevice.Clear(Color.
                                MediumSpringGreen);
    graphics.GraphicsDevice.BeginScene();
    // TODO: Add your drawing code here
    spriteBatch.Begin();
    spriteBatch.Draw(myTexture, new
            Rectangle(spriteX, spriteY, myTexture.Width,
            myTexture.Height), Color.TransparentWhite);
    spriteBatch.End();
    // Let the GameComponents draw
    DrawComponents();
    graphics.GraphicsDevice.EndScene();
    graphics.GraphicsDevice.Present();
}
```

Da notare l'utilizzo dell'oggetto spriteBatch istanziato durante il caricamento delle risorse: questo dovrà preparare la scena (metodo Begin), disegnare gli sprite in una determinata posizione e con l'opportuno sfondo (metodo Draw) ed infine chiudere la scena (meotodo End).
Con queste poche istruzioni avremo il nostro primo sprite andare in giro per lo schermo: al giorno d'oggi un semplice esercizio ma, vista la semplicità, ci permette di capire quanti passi avanti siano stati fatti da quando si doveva conoscere l'Hardware come le proprie tasche ed il C++ (se non l'assembler) come seconda lingua.

Fig. 1: ***Sprite a zonzo per lo schermo***

Il mondo dei videogiochi s'è evoluto nella terza dimensione: è proprio qui che XNA dà il meglio di se.

APPLICAZIONE DI ESEMPIO 3D

Prima di concentrarci sulla nostra applicazione di esempio 3D, è opportuno chiarire che XNA e' largamente basato su Managed DirectX ossia la versione di DirectX per .NET.
Una differenza fondamentale rispetto a Managed DirectX (precisamente stiamo parlando del componente Direct3D) è che la fixed-function pipeline è stata del tutto eliminata per dare spazio ad un modello di rendering shader-centrico ed anzi interamente basato su shaders.
Per comprendere la portata di un tale passaggio ci conviene fare un passo indietro, nei tempi dell'introduzione degli accelatori 3D nel mondo dei videogames.
Le prime schede grafiche supportavano determinate operazioni, atte ad accelerare in hardware una o più fasi del processo di rendering, che vanno sotto il nome di fixed-functions: Il motivo è dettato dal fatto che operazioni come ad esempio l'illuminazione di oggetti in una scena 3D, può essere effettuato solamente usando algoritmi ed equazioni prefissate.
D'altro canto l'introduzione degli shaders ha permesso di modificare e personalizzare il processo di rendering in dipendenza delle esigenze dell'applicazione sulla quale ci troviamo a lavorare: non solo è stato così possibile implementare in hardware molte più tecniche e metodi in maniera efficiente ma anche dar vita a nuova ricerca in computer grafica facente perno su shaders, chip grafici programmabili e così via. Direct3D supporta dei profili di rendering che sostanzialmente coincidono colla generazione di scheda grafica a cui facciamo riferimento: parleremo pertanto di Pixel o Vertex Shader 1.1, 2.0 o 3.0 per esempio.
XNA richiede una scheda grafica che supporti quantomeno VS e PS 1.1 per il semplice funzionamento ma Microsoft raccomanda una scheda basata su PS & VS 2.0 come entry-level.
Da un punto di vista pratico per fruire di tutte le features messe a disposizione da XNA sarebbe opportuno disporre di una scheda grafica che supporti interamente Direct3D 9.0c oltre a PS & VS 3.0 (ovviamente questo sarà automaticamente vero sulla Xbox 360).
La nostra semplice applicazione di esempio contiene alcuni elementi di basi utili in svariati contesti: oltre a mostrare come sia possibile creare una geometria minimale e come applicare un semplice shader, essa permette anche di controllare un semplice meccanismo di camera in terza persona.
Teniamo a chiarire che dal momento che XNA GSE non contiene ancora classi e metodi atti ad implementare la Content Pipeline, abbiamo preferito creare un esempio con della geometria davvero semplificata piuttosto che usare metodi lunghi e pochi chiari che carichino modelli in formato binario dal disco.
Per tutti coloro che fossero interessati ad avere maggiori dettagli in materia, basta dare uno sguardo al blog del team di XNA(http://blogs.msdn.com/xna/archive/2006/08/29/730168.aspx).
Passiamo quindi alla struttura della nostra applicazione: In primo luogo diamo un'occhiata alla dichiarazione della classe principale.

```
namespace WindowsGame
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    partial class CameraDemo : Game
    {
        // Riferimenti al componente grafico ed agli
        //              oggetti necessari per il rendering
        GraphicsComponent graphicsComponent;
        VertexDeclaration decl = null;
        VertexPositionColor[] vpc = null;
        // Parametri e riferimenti agli shader
        CompiledEffect compiledEffect;
        Effect effect;
        float delta = 0.0f;
        float deltainc = 0.25f;
      [...]
```

In buona sostanza viene dichiarato un namespace all'interno del quale è dichiarata una classe *CameraDemo* che eredita dalla classe Game inclusa nel framework di XNA.
Vengono dichiarate delle variabili membro usate all'interno, la cui lista completa è preferibile omettere per il momento per motivi di leggibilità. Proseguendo, troviamo il metodo OnStarting:

```
protected override void OnStarting()
{
            base.OnStarting();
            SetupShader();
            InitVertexBuffers();
            IsFixedTimeStep = false;

}
```

Dopo una chiamata al metodo *OnStarting* della classe base, questo metodo si incarica di caricare gli shader da file e compilarli.

```
void SetupShader()
{
   GraphicsDevice device =
                      graphicsComponent.GraphicsDevice;
   compiledEffect =
Effect.CompileEffectFromFile(".\\FX_Files\\ReallySimple
          Effect.fx", null, null,  CompilerOptions.Debug |
                         CompilerOptions.SkipOptimization,
                                  TargetPlatform.Windows);
   effect = new Effect(device,
                          compiledEffect.GetShaderCode(),
                              CompilerOptions.None, null);
}
```

Il metodo *InitVertexBuffers* invece si occupa di creare il buffer dei vertici che verranno usati per rendirizzare le geometria su schermo.
In questo caso stiamo creando un insieme di triangoli i cui unici attributi saranno posizione e colore, in maniera tale che i nostri shader possano poi provveddere a renderizzarli opportunamente.
Lo shader d'esempio accluso al codice effettua una dissolvenza verso il nero sul colore dei triangoli.

```
void InitVertexBuffers()
{
   GraphicsDevice device =
                      graphicsComponent.GraphicsDevice;
   decl = new VertexDeclaration(device,
                     VertexPositionColor.VertexElements);
   vb = new VertexBuffer(device,
                          typeof(VertexPositionColor), 3,
     ResourceUsage.WriteOnly, ResourcePool.Managed);
   vpc = new VertexPositionColor[3];
   vpc[0].Position = new Vector3(0, 0, 0);
   vpc[1].Position = new Vector3(0, 3, 0);
   vpc[2].Position = new Vector3(3, 0, 0);
   vpc[0].Color = Color.Red;
   vpc[1].Color = Color.Green;
   vpc[2].Color = Color.Blue;
   vb.SetData<VertexPositionColor>(vpc);
}
```

Continuando la nostra carrellata sul codice, abbiamo il metodo di Update ed i metodi ad esso correlati.

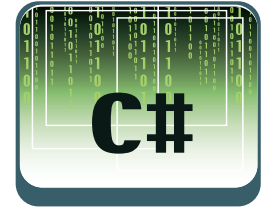
```
protected override void Update()
{
         base.Update();
         rotationSpeed = 1f *
          (float)ElapsedTime.TotalSeconds;
         forwardSpeed = 25f *
          (float)ElapsedTime.TotalSeconds;
```

```
         delta += deltainc *
                         (float)ElapsedTime.TotalSeconds;

         if (delta >= 1.0f)
         {
            delta = 1.0f;
            deltainc *= -1f;
         }
         else if (delta <= 0.0f)
         {
            delta = 0.0f;
            deltainc *= -1f;
         }

         UpdateComponents();
         GetKeyboardInput();
         UpdateAvatarPosition();
}
```

Dopo aver chiamato il metodo di Update della classe base, vengono ricalcolati delle variabili membro utilizzate internamente alla classe, rispettivamente la velocità di rotazione, traslazione e l'incremento (o decremento) di un fattore di fade del colore.
Successivamente, vengono aggiornati componenti software registrati con la classe (in questo caso nessuno), viene letto l'input da tastiera (per muovere la camera od uscire dall'applicazione premendo ESC), ed infine aggiornare gli attributi della camera.
Sorvoliamo sui dettagli implementativi della funzione di aggiornamento della posizione della camera in quanto piuttosto lunga ed intenta ad effettuare diverse operazioni di natura matriciale.
Il codice accluso contiene dei commenti a cui fare riferimento qualora si fosse interessati a saperne di più. Per concludere si passa al metodo che effettua il rendering vero e proprio, ossia Draw.

```
protected override void Draw()
{
      if (!graphicsComponent.EnsureDevice())
return;
      base.Draw();
      DrawComponents();
      DrawFrame();
}
```

Prima ci assicuriamo che il Device di rendering sia utilizzabile e successivamente passiamo al rendering: prima chiamiamo il metodo Draw della classe base, poi renderizziamo eventuali componenti registrati con la nostra classe ed infine renderizziamo un frame.

```
void DrawFrame()
{
 GraphicsDevice device =graphicsComponent.
                              GraphicsDevice;
```

FABIO ZAMBETTA
e' lettore presso la Scuola di Computer Science & IT della RMIT University, a Melbourne (Australia), dove insegna games programming e games design. Fabio Zambetta è laureato e dottorato in Informatica (conseguiti presso l'Università degli Studi di Bari), e' membro del comitato di standardizzazione per l'Intelligenza Artificiale nei videogames (AIISC) ed i suoi interessi di ricerca vertono su Interactive Storytelling, Intelligenza Artificiale nei videogames ed animazione facciale ed umana.

IGOR ANTONACCI
Lavora su ambienti Microsoft da circa 8 anni, passando da COM, COM+ passando da .NET fino all'attuale .NET 3.0 . Parallelamente ha avuto qualche esperienza con la programmazione a basso livello. Attualmente si occupa di sviluppo di sistemi Multi-tier Distribuiti ed occupa la posizione di Software Engineer all'interno di una Multinazionale del settore dell'Healthcare.

```
device.Clear(ClearOptions.Target |
ClearOptions.DepthBuffer, new Vector4(0, 0, 0, 255),
                                                  1.0f, 0);
    device.BeginScene();
        // Turn off culling so the back side of the
        // triangles can be seen
graphicsComponent.GraphicsDevice.RenderState.CullM
                                   ode = CullMode.None;
    UpdateCameraThirdPerson();
    DrawTriangles();
    device.EndScene();
    device.Present();
}
```

Il metodo DrawFrame si occupa prima di tutto di pulire il framebuffer. Le chiamate di rendering sono racchiuse tra la chiamata al metodo *BeginScene* ed il metodo *EndScene* della classe *GraphicsDevice* (l'istanza specifica nel codice si chiama *device*).
Il metodo UpdateCameraThirdPerson si occupa di effettuare il setup delle matrci di camera e di vista usate da XNA, mentre DrawTriangles effettua la vera e propria chiamata di rendering dei buffer di vertici, come riportato piu' in basso.

```
void DrawTriangles()
{
        GraphicsDevice device =
                    graphicsComponent.GraphicsDevice;
        device.VertexDeclaration = decl;
        for (int z = 0; z < 5; z++)
        {
            for (int x = 0; x < 5; x++)
            {
                world = Matrix.CreateTranslation(new
                                Vector3(x * 3, 0, z * 3));
                combinedMatrix = world * view * proj;
effect.Parameters["WorldViewProj"].SetValue(combined
                                                Matrix);
effect.Parameters["delta"].SetValue(delta);
                effect.CurrentTechnique =
                effect.Techniques["TransformTechnique"];
effect.Begin(EffectStateOptions.Default);
                foreach (EffectPass pass in
                        effect.CurrentTechnique.Passes)
                {
                    pass.Begin();
device.DrawUserPrimitives<VertexPositionColor>(Primi
                        tiveType.TriangleList, 1,vpc);
                    effect.CommitChanges();
                    pass.End();
                }
                effect.End();
            }
        }
}
```

Il metodo è abbastanza complesso in quanto XNA sta utilizzando un effetto implementato da vertex e pixel shader caricati in precedenza.
Come è possibile notare la chiamata a DrawUserPrimitive è preceduta dal setup dei parametri necessari al rendering (matrici di camera vista e parametri dello shader), e racchiuse tra chiamate di *Begin ed End* per l'effetto stesso.

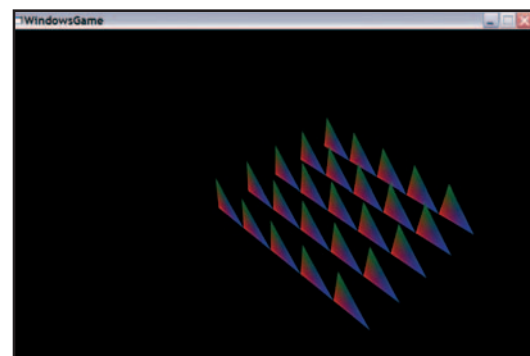



Fig. 2: *Triangoli ballerini*

Per un'introduzione all'argomento shader ed effetti in DirectX consigliamo la lettura di "DirectX 9 Programmable Graphics Pipeline", ad opera di Kris Gray per Microsoft Press, stampato nel 2003.
Il risultato finale e' riportato nello schermata piu' in basso.

CONCLUSIONI

La produzione di videogames è entrata a far parte (purtroppo per i romantici) di quel genere di software a produzione "industriale" ovvero con metodologie e strumenti ben collaudati; daltronde i ritmi di uscita sono sempre più incalzanti e la produzione "artigianale" (come quella che avveniva fino a pochi anni or sono) era impraticabile.
Microsoft ha posto l'ennesima pietra miliare in questo, portando agli sviluppatori (professionisti e non) uno strumento ancora più facile e immediato e che astrae ancor di più da tediosi dettagli tecnici.
XNA è, comunque, ancora un prodotto acerbo (nella sua Beta1) e mancano componenti fondamentali come la "Content Pipeline" ovvero quella parte di Framework che si occupa di caricare modelli 3D, texture in formati diversi e tracce audio per poi trasformarli ed inserirli nel ciclo di vita del gioco.
Sicuramente le Beta successive (ed ancor di più la versione finale) ci permetteranno di saltare anni luce avanti rispetto ai tempi in cui ci si doveva calcolare i cicli di CPU di ogni singola istruzione Assembly.

Fabio Zambetta, Igor Antonacci

UN GESTORE PER LE MAILING LIST

IN QUESTO ARTICOLO REALIZZEREMO UNA WEB APPLICATION IN GRADO DI CONTROLLARE L'INTERO CICLO DI VITA DI UN SERVIZIO DI DISTRIBUZIONE DI POSTA ELETTRONICA MEDIANTE LISTA. SARÀ NECESSARIO USARE TECNICHE DECISAMENTE INNOVATIVE..

A volte è necessario, programmando per il web, realizzare applicazioni che non si esauriscono nello scambio "richiesta risposta". Per esemplificare questo genere di applicazioni esaminiamo il caso di un servizio di Mailing List.Il servizio che dovremmo realizzare dovrà avere i seguenti requisiti minimi:

- Gestione di aggiunta e cancellazione di categorie (argomenti dei messaggi).
- Gestione di aggiunta e cancellazione utenti con aggiunta, modifica e cancellazione delle categorie a cui gli utenti possono essere iscritti.
- Gestione di aggiunta e cancellazione di messaggi
- Processo ciclico di invio dei messaggi agli utenti

L'ultimo requisito dell'applicazione presenta la difficoltà maggiore: tipicamente un'applicazione di mailing list resta attiva in background e, periodicamente controlla se ci sono nuovi messaggi da inoltrare alla lista di utenti.
Per implementare un'applicazione di questo genere la via maestra sarebbe quella di sviluppare un servizio di Windows che possa interagire con la parte amministrativa.
Tuttavia, seguendo questa strada, l'applicazione non funzionerebbe in uno sito collocato presso un Internet Service Provider.
Moltissimi sviluppatori Web, infatti, non hanno il controllo del server di produzione: hanno a disposizione uno spazio presso un ISP e basta, ben difficilmente il Provider permetterà di installare software aggiuntivo, come appunto un Servizio di Windows, che potrebbe anche mandare in crash il server.
A tutto questo aggiungiamo un'altra difficoltà in più : oggi sviluppiamo l'applicazione con interfaccia Web, ma in un ipotetico futuro potrebbe essere necessario sviluppare un client desktop per gestirla.
Come fare quindi a soddisfare tutti questi, non indifferenti, requisiti?

La soluzione sta nell'utilizzo di qualcosa di intermedio tra l'applicazione vera e propria e i vari client, e cosa c'è di più standard per fare questo dei Webservices (fig.1)?

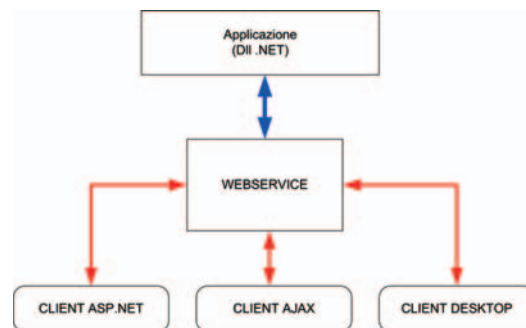



Fig. 1: *Schema dell'applicazione*

PREPARIAMO L'AMBIENTE DI LAVORO

Iniziamo quindi a preparare il nostro ambiente di lavoro in Visual Studio 2005.
Per prima cosa, per garantirci una migliore flessibilità, creiamo una soluzione vuota (fig.2)Alla soluzione aggiungiamo poi un progetto di tipo Libreria che conterrà tutta la logica applicativa e quindi un progetto web che conterrà i nostri webservices.

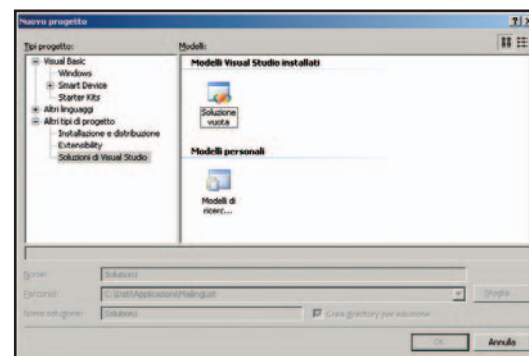



Fig. 2: *Creazione della Soluzione in VS 2005*

Per testare, in fase di sviluppo, il funzionamento dell'applicazione (che, lo ricordiamo, è contenuta nella libreria) possiamo poi aggiungere anche un progetto di applicazione console che ci tornerà utile per provare il funzionamento delle singole parti dell'applicazione.
Alla fine la soluzione dovrebbe essere popolata come in **fig. 3**.

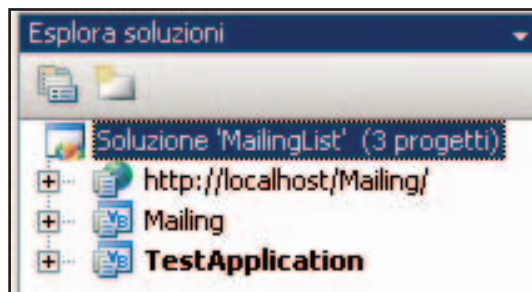



Fig. 3: ***La Soluzione completa con i vari progetti***

LA LIBRERIA

La libreria, che chiameremo appunto MailingList, è il cuore della nostra Applicazione, i webservices saranno infatti semplicemente il mezzo per esporne le funzionalità.
In primo luogo definiamo quindi gli oggetti che devono contenere i dati che ci servono:

- Categorie
- Utenti
- Messaggi

Infatti la nostra Mailing List sarà categorizzabile, un utente potrà quindi iscriversi ad una o più categorie e ricevere solo i messaggi che gli interessano. Ogni oggetto **Utente** conterrà quindi una lista di categorie. Anche ogni oggetto **Messaggio** contiene il riferimento ad una lista di categorie, in modo che, in fase di invio, si potrà recuperare la lista di utenti a cui effettuare l'inoltro tra tutti gli utenti che hanno le categorie del messaggio nella propria lista di categorie.
Il **Messaggio** dovrà avere però anche un **Registro** associato dove il programma può scrivere l'esito di ogni tentativo di invio, in modo da evitare di inviare più volte lo stesso messaggio ad un utente. Al fine di evitare ripetizioni di proprietà comuni e di mantenere l'unitarietà dei dati si creeranno poi delle classi astratte dalle quali far discendere, a cascata i vari oggetti concreti. Tali classi saranno : InnerObject, che rappresenta tutti gli oggetti di dati che creeremo, e MailingObject , discendente a sua volta da InnerObject, che fornisce alcune proprietà comuni.Questo modello dei oggetti di dati è rappresentato in **fig.4**.

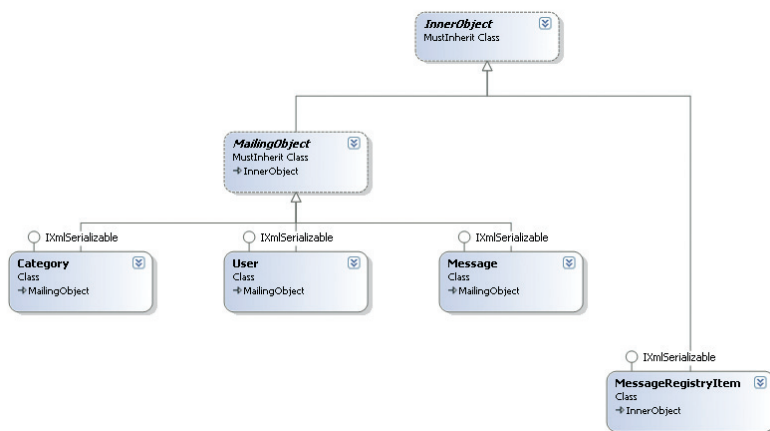



Fig. 4: ***Il modello degli oggetti dati***

LE LISTE

Ma oltre agli oggetti che rappresentano le singole unità di dati (Categoria,Utente, Messaggio,Voce del registro messaggi) dovremo anche creare le Liste che li rappresentano.
In particolare per le liste di Categoria,Utente e Messaggio abbiamo scelto un criterio analogo a quello seguito per gli oggetti dati, ovvero di implementare una classe astratta che rappresenta una lista di oggetti MailingObject (utilizzando i Generics) dalla quale far derivare le liste di oggetti corrispondenti **fig. 5**.

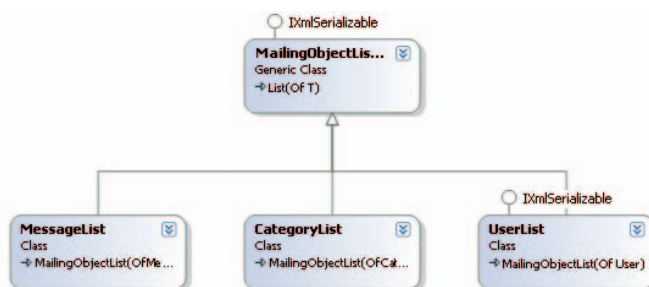



Fig. 5: ***Il modello ad oggetti delle liste***

Per completare questo primo quadro di insieme, occorre citare la classe che si occuperà, in modo unitario, della gestione dei dati ovvero Manager. Sarà quest'oggetto a rappresentare l'interfaccia verso i vari webservices fornendo anche i metodi per il funzionamento vero e proprio dell'applicazione.

LA SERIALIZZAZIONE DEI DATI

Sulla serializzazione e la differenza tra la modalità XML e Binaria si segnala il link:
http://www.15seconds.com/issue/020903.htm

LA PERSISTENZA DEI DATI

Prima di analizzare il codice parliamo del modello che abbiamo scelto per realizzare la persistenza dei dati.
Il framework .NET offre fondamentalmente due

modelli di serializzazione (ovvero la possibilità di salvare un oggetto in un file per poi poterlo ricostruire da esso) : la serializzazione XML e la serializzazione Binaria. La serializzazione XML conserva la leggibilità esterna dei dati ma produce files più grandi, la serializzazione binaria, per contro, produce files di dimensioni più contenute ma che restano utilizzabili solamente dall'applicazione che li ha prodotti.
Nella nostra applicazione abbiamo scelto di lasciare aperte tutte e due queste possibilità, nella classe **Manager** (che contiene tutti i nostri oggetti) abbiamo infatti dapprima definito una matrice di estensioni associate alla serializzazione XML:

```
<NonSerialized()> Private Shared XmlExtensions()
            As String = New String() {".xml", ".mml"}
```

Nel successivo metodo Save la serializzazione avviene in modo XML o Binario a seconda se l'estensione del nome del file in cui serializzare l'oggetto appartenga o meno alla matrice *XmlExtensions* :

```
    Private filename As String
    Public Sub Save()
        ...
            Dim ext As String =
                IO.Path.GetExtension(filename).ToLower
            If Array.IndexOf(Of String)(XmlExtensions,
                                        ext) <> -1 Then
                XmlSerialize(Me)
            Else
                BinarySerialize(Me)
            End If
        ...
    End Sub
```

Naturalmente vengono poi definiti i metodi per serializzare nei due modi:

```
    Private Shared Sub XmlSerialize(ByVal man As
                                          Manager)
        Dim fs As IO.FileStream = Nothing
            Dim ser As New
                     XmlSerializer(GetType(Manager))
            fs = New IO.FileStream(man.filename,
                IO.FileMode.Create, IO.FileAccess.Write)
            ser.Serialize(fs, man)
    ...
    End Sub
    Private Shared Sub BinarySerialize(ByVal man As
                                          Manager)
        Dim fs As IO.FileStream = Nothing
            Dim ser As New
              System.Runtime.Serialization.Formatters.
                          Binary.BinaryFormatter()
            fs = New IO.FileStream(man.filename,
                IO.FileMode.Create, IO.FileAccess.Write)
            ser.Serialize(fs, man)
    ...
    End Sub
```

Ma perché utilizzare dei semplici files e non un ben più robusto database?
Molto a dir la verità dipende dalle esigenze di scalabilità dell'applicazione: se dovessimo pensare di avere migliaia di utenti, decine di categorie e centinaia di migliaia di messaggi ovviamente non potremmo fare a meno di utilizzare un robusto Database Relazionale.
Tuttavia queste condizioni sono spesso marginali: recentemente mi è capitato di fare un confronto tra la deserializzazione da file ed il recupero diretto di dati contenuti in circa 30000 righe di una tabella di database ed ho potuto verificare come con la deserializzazione i tempi si riducano al 10% di quelli occorrenti con la classica connessione.
Il fatto è che molto frequentemente le applicazioni web utilizzano i database come supporto per i dati anche quando la quantità di quest'ultimi è relativamente ridotta: il tempo tra connessione, esecuzione della query e restituzione dei dati risulta quindi molto superiore anche rispetto ad una semplice lettura di un file di testo.
Nel nostro caso quindi è più che sufficiente ricorrere alla sola serializzazione, tanto più che la classe Manager prevede un metodo per archiviare in un altro file i vecchi messaggi:

```
Public Function ArchiveMessages(ByVal
  archiveFilename As String, ByVal archiveErrors As
    Boolean, ByVal minDate As DateTime) As String
        Dim mng As Manager =
                     Manager.Create(archiveFilename)
        mng.Users = Me.Users
        mng.Categories = Me.Categories
        Dim msgToRemove As New List(Of Message)
        For Each msg As Message In Me.Messages
            Dim archiveThis As Boolean
            If archiveErrors Then
                archiveThis = (msg.DeliveryStatus =
  DeliveryStatus.completed Or msg.DeliveryStatus =
    DeliveryStatus.errors) And msg.MessageDate <=
                                             minDate
            Else
                archiveThis = msg.DeliveryStatus =
      DeliveryStatus.completed And msg.MessageDate
                                        <= minDate
            End If
            If archiveThis Then
                mng.Messages.Add(msg)
                msgToRemove.Add(msg)
```

```
            End If
        Next
        mng.OnChanges()
        If msgToRemove.Count > 0 Then
            For Each remMsg As Message In
                                        msgToRemove
                Me.Messages.Remove(remMsg)
            Next
        End If
        Me.OnChanges()
        Return mng.filename
    End Function
```

ENTRIAMO NEL DETTAGLIO

Dopo aver visto, per sommi capi, il modello di base utilizzato (gli attori della nostra scena) scendiamo un po' in dettaglio per vedere i punti salienti del codice.
Partiamo quindi dalla classe comune a tutti gli oggetti di dati utilizzati, **InnerObject**, che si presenta semplicemente come:

```
Public MustInherit Class InnerObject
    Private _Owner As Manager
    Public Property Owner() As Manager
        Get
            Return _Owner
        End Get
        Set(ByVal Value As Manager)
            _Owner = Value
        End Set
    End Property
End Class
```

Essendo una classe molto generale **InnerObject** espone solo una proprietà che sarà comune a tutti gli oggetti discendenti : *Owner* ovvero il **Manager** al quale appartiene l'oggetto.
Un po' più dettagliata è la classe **MailingObject** che, come abbiamo visto, è comune a Categorie,Utenti e Messaggi :

```
Public MustInherit Class MailingObject
    Inherits InnerObject

    Private _UniqueID As Integer
    Public Property UniqueID() As Integer
        ...
    End Property
    Private _Name As String
    Public Property Name() As String
        ...
    End Property
    Public Function GetProperty(ByVal propName As
                                        String) As Object
        Dim pInfo As PropertyInfo
        pInfo = Me.GetType.GetProperty(propName,
        BindingFlags.Instance Or BindingFlags.Public Or
BindingFlags.IgnoreCase Or BindingFlags.NonPublic)
        If pInfo IsNot Nothing AndAlso pInfo.CanRead
                                                Then
            Return pInfo.GetValue(Me, Nothing)
        End If
        Return Nothing
    End Function
    Public Sub SetProperty(ByVal propName As
                        String, ByVal value As Object)
        Dim pInfo As PropertyInfo
        pInfo = Me.GetType.GetProperty(propName,
        BindingFlags.Instance Or BindingFlags.Public Or
BindingFlags.IgnoreCase Or BindingFlags.NonPublic)
        If pInfo IsNot Nothing AndAlso
                                pInfo.CanWrite Then
            pInfo.SetValue(Me, value, Nothing)
        End If
    End Sub
    Public Sub SetProperties(ByVal properties() As
        String, ByVal values() As String, Optional ByVal
                    raiseExceptions As Boolean = False)
        If properties Is Nothing Then Return
        If values Is Nothing Then Return
        For i As Integer = 0 To properties.Length - 1
            If Not String.IsNullOrEmpty(properties(i))
                        AndAlso values.Length > i Then
                Try
                    SetProperty(properties(i), values(i))
                Catch ex As Exception
                    If raiseExceptions Then
                        Throw ex
                    End If
                End Try
            End If
        Next
    End Sub
    Public Sub SetProperties(ByVal propertiesList As
    String, ByVal valuesList As String, Optional ByVal
        separator As String = ";", Optional ByVal
                    raiseExceptions As Boolean = False)
        Dim properties() As String =
                        propertiesList.Split(separator)
        Dim values() As String =
                            valuesList.Split(separator)
        SetProperties(properties, values,
                                        raiseExceptions)
    End Sub
End Class
```

Come proprietà comuni ha solo l'ID e il Nome (*UniqueID e Name*) ma dispone anche di una serie di metodi utili a recuperare ed impostare mediante Reflection le proprietà dell'oggetto erede.

NOTA

IL CODICE ALLEGATO

Il codice allegato nel CD è pressoché completo per la parte dei Webservices e della libreria. Per esigenze di spazio nell'articolo non possiamo entrare troppo nel dettaglio del funzionamento di tutte le classi. Nel codice vi sono tuttavia commenti esplicativi. Per il funzionamento del servizio di recapito modificare opportunamente la sezione appSettings del file web.config

In tal modo, ad esempio, l'oggetto User che eredita da MailingObject ed ha una proprietà di nome *Mail* per impostarla potrà usare sia la sintassi tradizionale:

```
Dim u as new User
u.Mail="prova@email.com"
```

Oppure:

```
u.SetProperty("Mail","prova@email.com")
```

Mentre per impostare, in una volta sola, più proprietà potremmo usare:

```
u.SetProperties("UniqueId;Name;Mail","1;Antonio
                                   Bianchi; prova@email.com")
```

Ciò si rivelerà utile nei metodi dei Webservices dove, anziché usare metodi con molti parametri come:

```
<WebMethod> _
Function AddUser(id As Integer,Name As String,Mail
                  As String,Password As String) As User
```

Sarà possibile ridurre i parametri a due:

```
<WebMethod> _
Function AddUser(propertiesList As String,
                          valuesList As String) As User
```

Il che risulta particolarmente comodo in caso di chiamate da AJAX Un altro cenno merita poi la classe **MailingObjectList**, che è quella che fa da base alle liste di oggetti MailingObject :

```
Public Class MailingObjectList(Of T As
                  {MailingObject, New, IXmlSerializable})
    Inherits List(Of T)
    Implements IXmlSerializable
    Protected Function GetList(ByVal query As String,
        ByVal order As String, ByVal propertyMaps As
                              String) As List(Of T)
…
    End Function
    Protected Function GetList(ByVal query As String,
        ByVal order As String, ByVal propertyMaps As
        String, ByVal start As Integer, ByVal count As
                              Integer) As List(Of T)
        …
    End Function
…
    Public Overloads Sub Add(ByVal Item As T)
        MyBase.Add(Item)
        If Item.UniqueID = 0 Then
            Item.UniqueID = LastId + 1
            LastId += 1
        End If
    End Sub
    Public Function CreateItem(ByVal name As
                                    String) As T
        …
    End Function
    Protected Overloads Function GetRange(ByVal
            UniqueIds() As Integer) As List(Of T)
        …
    End Function
    Public Function GetItemById(ByVal UniqueId As
                                  Integer) As T
…
    End Function
    Public Sub RemoveById(ByVal UniqueId As
                                    Integer)
        …
    End Sub
    Public Overloads Function FindAll(ByVal
  propertiesList As String, ByVal valuesList As String,
 Optional ByVal separator As String = ";") As List(Of
                                              T)
        …
    End Function
    Protected Friend Sub Fill(ByVal list As List(Of T),
     Optional ByVal eraseContent As Boolean = False)
        …
    End Sub
    Protected Friend Function Fill(Of R As {New,
 MailingObjectList(Of T)})(ByVal list As List(Of T)) As
                                              R
        …
    End Function
    Public Function GetItems(ByVal listIds As List(Of
                          Integer)) As List(Of T)
        Dim result As New MailingObjectList(Of T)
        For Each member As T In Me
            If listIds.Contains(member.UniqueID) Then
                result.Add(member)
            End If
        Next
        Return result
    End Function
    Private _LastId As Integer
    Public Property LastId() As Integer
…
    End Property
End Class
```

In particolare qui troviamo i metodi per aggiungere un ID progressivo agli oggetti, per ottenere liste filtrate attraverso la Reflection, per caricare la lista con gli oggetti di un'altra lista e per fornire una rappresentazione serializzata della lista stessa.

Non abbiamo qui lo spazio per analizzare nel dettaglio anche la classe Manager (che trovate, come tutto il codice di cui parliamo, nel CD alle-

gato), tuttavia dobbiamo soffermarci su alcuni punti.
In primo luogo Manager ha, come proprietà, le liste di Categorie,Utenti e Messaggi:

```
Private _Users As UserList
Public Property Users() As UserList
    Get
        If _Users Is Nothing Then
            _Users = New UserList(Me)
        End If
        Return _Users
    End Get
    Set(ByVal Value As UserList)
        _Users = Value
    End Set
End Property
Private _Categories As CategoryList
Public Property Categories() As CategoryList
…
End Property
Private _Messages As MessageList
Public Property Messages() As MessageList
…
End Property
```

E quindi, fondamentale, il metodo Process Messages, che verrà ripetuto ciclicamente, che processa i messaggi:

```
Public Sub ProcessMessages()
    For Each msg As Message In Me.Messages
        If msg.DeliveryStatus <>
                    DeliveryStatus.completed Then
            Dim hasErrors As Boolean = False
            For Each regItem As
                MessageRegistryItem In msg.Registry
                If regItem.Status <>
                    DeliveryStatus.completed Then
                    Dim u As User =
        Me.Users.GetItemById(regItem.User.UniqueID)
                    Try
                        Sendmail.Send(u.Mail,
                          msg.Name, msg.Body, False)
                        regItem.LastError = ""
                        regItem.Status =
                              DeliveryStatus.completed
                    Catch ex As Exception
                        hasErrors = True
                        regItem.LastError =
                                          ex.Message
                        regItem.Status =
                                 DeliveryStatus.errors
                    End Try
                    regItem.ItemDate =
                                        DateTime.Now
                End If
            Next
            If Not hasErrors Then
                msg.DeliveryStatus =
                        DeliveryStatus.completed
            Else
                msg.DeliveryStatus =
                            DeliveryStatus.errors
            End If
        End If
    Next
    Threading.Thread.Sleep(5000)
    ProcessMessages()
End Sub
```

La versione che riportiamo qui è molto ridotta rispetto a quella presente nel codice (che gestisce anche il logging e una serie di possibili eccezioni), vengono solo riportate le operazioni essenziali che sono:

- Scorrere tutti i messaggi
- Per ogni messaggio in cui lo stato non sia completato scorrere tutte le voci del Registro
- Recuperare l'utente e provare a inviare il messaggio
- In caso di esito positivo impostare lo stato completed alla voce di registro
- In caso di esito positivo per tutti gli utenti impo stare lo stato completed del messaggio
- Aspettare 5 secondi e poi ripetere il ciclo

I WEBSERVICES

Completata la nostra libreria passiamo quindi a vedere come implementare i Webservices.
Per prima cosa c'è da definire una risposta standard che devono fornire tutti i metodi dei Webservices. Questo perché, tra le specifiche iniziali, avevamo anche il fatto di poter usare i Webservices da una pluralità di ambienti (ASP.NET, AJAX, Desktop) ed in questi casi la cosa migliore è definire un pattern costante per il colloquio con il server.
Ci spostiamo quindi nella cartella **App_Code** del progetto web della nostra Soluzione e creiamo un oggetto che rappresenterà lo standard di risposta di tutti metodi dei Webservices:

```
Public Class OperationResult(Of T As {New})
    Public Property Success() As Boolean
        …
    Public Property Message() As String
        …
    Public Property Data() As T
        …
    Public Shared Function Positive(Of R As
 {OperationResult(Of T), New})(ByVal data As T) As R
        Dim result As New R
```

```
        result.Success = True
        result.Data = data
        Return result
    End Function
    Public Shared Function Negative(Of R As
  {OperationResult(Of T), New})(ByVal msg As String)
                                              As R
        Dim result As New R
        result.Success = False
        result.Data = Nothing
        result.Message = msg
        Return result
    End Function
    Public Shared Function CheckFailed(Of R As
 {OperationResult(Of T), New})(ByVal checkResult As
                          OperationCheckResult) As R
        Select Case checkResult
            Case OperationCheckResult.NoAuth
                Return Negative(Of R)("Diritti
                                      insufficienti")
            Case OperationCheckResult.NoUser
                Return Negative(Of R)("Utente non
                                       riconosciuto")
        End Select
        Return Nothing
    End Function
End Class
```

L'oggetto generico OperationResult dispone di tre proprietà:

- Success – che comunica il successo ho meno della richiesta
- Message – che veicola eventuali messaggi di errore al client
- Data – che rappresenta l'oggetto del tipo su cui verrà instanziata la classe

Abbiamo poi tre metodi shared per creare l'oggetto modellato su una classe derivata di OperationResult (per ogni oggetto della libreria, Utente,Categoria ecc. è stata infatti creata una classe derivata di OperationResult) : per creare la risposta positiva, per creare una risposta negativa generica e per creare una risposta negativa di mancata autenticazione.
Si è poi proceduto a creare, in App_Code, una classe generale chiamata CommonService dalla quale deriveranno tutti i Webservices della nostra applicazione, in questa classe c'è un metodo, **DoOperation**, che consente di costruire mediante Reflection un OperationResult tipizzato, riportiamo solo l'intestazione:

```
Protected Function DoOperation(Of T As {New}, R As
  {OperationResult(Of T), New})(ByVal operatorId As
          String, ByVal targetObject As Object, ByVal
     methodName As String, ByVal operationName As
     String, ByVal ParamArray Arguments() As Object)
                                                As R
```

Dove R rappresenta l'OperationResult tipizzato impostato sul tipo T.
In questo modo siamo in grado di seguire lo stesso pattern per tutti i Webmethods, ad esempio riportiamo quello che ritorna l'OperationResult (nella sua variante UserListResult) che contiene la lista degli utenti, ovvero:

```
<WebMethod()> _
Public Function GetList(ByVal operatorId As String,
     ByVal query As String, ByVal order As String) As
                                        UserListResult
      Return DoOperation(Of UserList,
    UserListResult)(operatorId, Mailer.Users, "GetList",
                           "users.getlist", query, order)
End Function
```

Come vediamo i metodi esposti dai Webservices non sono che dei "segnaposto" per le effettive operazioni che vengono effettuate da **DoOperation** in modo da garantire l'uniformità del flusso di risposta.
Attraverso questo sistema si mappa l'interfaccia dei Webservices ai metodi della classe di gestione **Manager**, che è resa disponibile a tutta l'applicazione dichiarandola come proprietà Mailer in **CommonService**:

```
Private Shared _Mailer As Manager
    Public Shared Property Mailer() As Manager
        Get
            Return _Mailer
        End Get
        Set(ByVal Value As Manager)
            _Mailer = Value
        End Set
    End Property
```

Già, ma dov'è che si instanzia la proprietà **Mailer**? Ed ancora, dov'è che parte il metodo che processa i messaggi in background anche quando nessun utente è collegato?
Il trucco sta tutto nel file global.asax dove, nell'evento Application_Start (all'avvio dell'applicazione Web, quindi) instanziamo la proprietà Mailer e facciamo eseguire il suo metodo ricorsivo, ProcessMessages, in un Thread separato:

```
    Public Property BackgroundWorker() As Thread
        ...
    Sub Application_Start(ByVal sender As Object,
                                  ByVal e As EventArgs)
        Dim mailerFile As String
        ...
```

LA REFLECTION

La Reflection è la possibilità offerta dai linguaggi di alto livello (come C# e VB.NET) di esaminare ed eseguire propri componenti, per informazioni sulla Reflection in .NET un link è : http://msdn2.microsoft.com/it-it/library/cxz4wk15.aspx

```
        Dim filename As String =
                    Server.MapPath(mailerFile)
        CommonService.Mailer =
                    Manager.Create(filename)
        ...
        BackgroundWorker = New Thread(AddressOf
            CommonService.Mailer.ProcessMessages)
        BackgroundWorker.Start()
    End Sub
```

In questo modo avremmo un'istanza di Manager a disposizione di tutti i Webservices per la gestione di Categorie,Utenti, Messaggi ecc. mentre in un thread separato (senza influire quindi sul funzionamento dei Webservices) gira di continuo il metodo di invio dei messaggi.
In questo modo da quando l'applicazione Web si avvia (nel momento in cui almeno un utente si collega) a quando si ferma (quando IIS viene fatto ripartire oppure si apportano modifiche al codice) il servizio di recapito sarà sempre attivo.

TEST SU STRADA

Proviamo quindi il nostro servizio che, anche se privo di interfaccia, è già funzionante utilizzando le Url dei vari Webservices (che sono categories.asmx, users.asmx , messages.asmx). Sul servizio categories.asmx aggiungiamo una nuova Categoria, "Cultura", con il metodo AddItem ed otteniamo come risposta la lista delle categorie impostate (**fig. 6**).

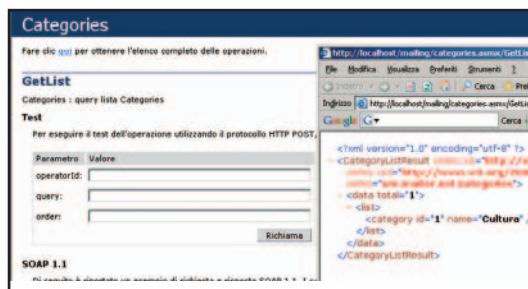



Fig. 6: ***Definiamo una categoria***

Sul servizio users.asmx impostiamo poi, analogamente, un nuovo utente al quale associamo la categoria 1, che corrisponde all'ID di "Cultura" (**fig. 7**).
Infine sul servizio messages.asmx imposteremo un nuovo messaggio diretto alla categoria 1 (**fig.8**). Mentre stavamo facendo questo c'era naturalmente il nostro Thread che stava lavorando in background per cui, dopo qualche secondo, vedremo recapitarci in Outlook il messaggio di prova (**fig. 9**)

Fig. 7: ***Aggiungiamo un utente***

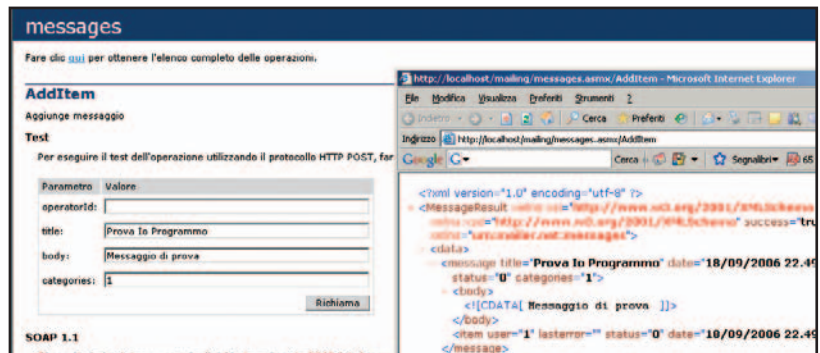



Fig. 8: ***Scriviamo un nuovo messaggio***

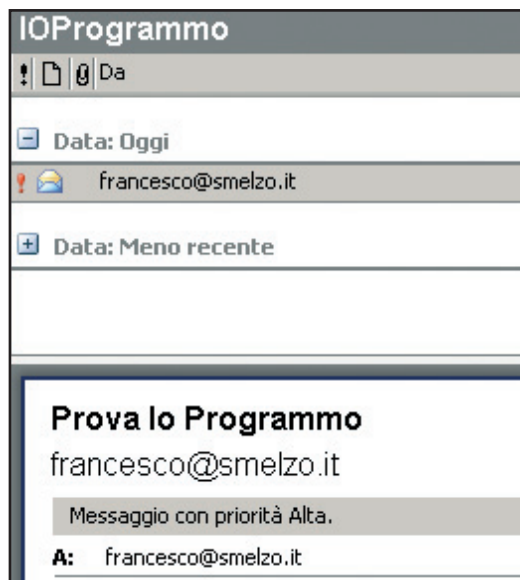



Fig. 9: ***Tutti gli header necessari***

CONCLUSIONI

Abbiamo visto come costruire un vero e proprio servizio di Mailing List sempre attiva utilizzando metodi e strumenti disponibili anche presso spazi di Internet Service Providers.
Fin qui ci siamo limitati a dare un'occhiata, seppur sommaria, all'implementazione del lato server con i Webservices.
In un prossimo numero invece proveremo a costruire una vera e propria interfaccia Web con AJAX e ASP.NET.

Francesco Smelzo

Francesco Smelzo è specializzato nello sviluppo in ambiente Windows con particolare riferimento ad applicazioni in ambiente .NET sia web-oriented che desktop.
Il suo sito web è www.smelzo.it.
Come sempre è a disposizione per ricevere suggerimenti o richieste sull'articolo all'indirizzo di posta elettronica
francesco@smelzo.it

AJAX SEMPLICE GRAZIE AD ECHO2

L'UTILIZZO DI QUESTO NUOVO FRAMEWORK PUÒ INCREMENTARE NOTEVOLMENTE LA FACILITÀ E LA VELOCITÀ DI SVILUPPO DI WEB APPLICATION E FA FARE A JAVA UN PASSO AVANTI NELLA DIREZIONE DEL WEB 2

REQUISITI

Conoscenze richieste
Java, Servlet e HTML

Software
JDK 1.4, Tomcat ed Eclipse 3.x

Impegno

Tempo di realizzazione

Negli ultimi tempi le tecnologie internet sono diventate così sofisticate da rendere possibile un avvicinamento delle applicazoni Web alla raffinatezza delle classiche interfacce standalone. Se lato server è da tempo che si utilizzano tecnologie consolidate, dal lato client il livello delle interfacce grafiche ottenuto con ambienti di sviluppo facili ed immediati come Delphi, Visual Basic, .NET e Java sembrava, fino a poco tempo fa, ancora iraggiungibile in un contesto web.
A prescindere dai tool utilizzati il limite maggiore è costituito dalla natura stessa delle applicazioni Web. Infatti esse sono basate su pagine html che ad ogni click dell'utente vengono scaricate interamente dal server. Questo comportamento determina, nel transitorio, la visualizzazione di una pagina vuota. La durata di tale transitorio dipende fortemente dalle condizioni della rete e dal peso in byte dei contenuti che si stanno scaricando.
Ajax è la tecnologia che permette di abbattere i limiti appena descritti. In questo articolo ci occuperemo del framework Echo2 rilasciato sotto licenza open source e scritto interamente in Java.
Echo2 permette la realizzazione di applicazioni Ajax senza conoscerne a fondo e nel dettaglio tutte le caratterisiche di questa tecnologia. Inizieremo a conoscerlo attraverso un esempio molto semplice: un "Helloworld". Successivamente ci caleremo nell'implementazione di un'applicazione un po' più complessa.

L'UNIONE FA LA FORZA

**Ajax è l'acronimo di Asynchronous Javascript and XML. In realtà è la combinazione delle seguenti tecnologie: XMLHttpRequest, Javascript, CSS e DOM (Document Object Model).
La vera novità risiede nell'XMLHttpRequest che, al di là del nome forse non proprio azzeccatissimo, è un oggetto messo a disposizione dai moderni browser per effettuare richieste asincrone verso il server. Quindi consente di effettuare richieste verso un server senza bloccare il flusso di esecuzione sul browser fino al caricamento della nuova pagina. Evitando la visualizzazione della pagina vuota, i contenuti vengono aggiornati solo quando la risposta giunge al browser. Inoltre Ajax facilita l'aggiornamento di limitate zone della pagina web; ciò è possibile con l'ausilio di Javascript, dei fogli di stile e di DOM, che consente con una sintassi Javascript di accedere ad elementi della pagina web.**

INSTALLARE ECHO2

Echo2 è un framework OpenSource realizzato da NextApp e rilasciato con licenza MPL 2.0; la versione 2.0 di Echo2 è scaricabile a partire dalla URL http://www.nextapp.com/platform/echo2/echo/download/. Scaricato il file compresso (.zip), supponiamo di scompattarlo sotto *C:* ottenendo la directory *C:\NextApp_Echo2*. A questo punto potremmo partire già con lo sviluppo della nostra prima applicazione, però vogliamo migliorare il framework con una libreria di componenti molto accattivanti scaricabili alla *URL http://sourceforge.net/project/showfiles.php?group_id=57452*. Scaricata la libreria, allo stato attuale alla versione 2.1.rc3, possiamo scompattarla sotto la directory *C:\NextApp_Echo2* ottenendo la directory *C:\NextApp_Echo2\echopointng* . I due pacchetti contengono sorgenti, documentazione e, come vedremo da qui a poco, i jar delle librerie già pronti da essere inclusi nei classpath delle nostre applicazioni.

HELLO WORLD ECHO2

E' giunto il momento di vedere all'opera questo fantastico framework e lo faremo implementando una banale applicazione che ce ne mostrerà le potenzilità. La nostra applicazione sarà composta da una Label, contenete un messaggio testuale, e da un link html alla cui pressione viene modificato l'aspetto della precedente Label. Come tutte le applicazioni web scritte in Java il punto di partenza sarà una particolare Servlet

```
public class ClientServlet extends WebContainerServlet {
        public ApplicationInstance
                        newApplicationInstance() {
                        return new Client();
        }
}
```

La nostra classe estende la classe astratta *WebContainerServlet* ed è costretta ad implementare il metodo *newApplicationInstance*. Questo metodo restituisce un oggetto istanza della classe *ApplicationInstance* estesa come vedremo dalla classe *Client*.

```
public class Client extends ApplicationInstance {
  private int i=0;
  private Label label=null;

  public Window init() {
    Window window = new Window();
    ContentPane contentPane = new ContentPane();
    window.setContent(contentPane);
    Column column=new Column();
    label = new Label("Hello Echo2: numero="+i);
    Button link=new Button("Incrementa numero");
    link.setForeground(Color.BLUE);
    link.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent ev) {
        i++;
        label.setText("Hello Echo2: numero="+i);
      }
    });
    column.add(label);
    column.add(link);
    contentPane.add(column);
    return window;
  }
}
```

Come possiamo notare lo stile con cui è implementata questa classe ricorda molto quello che appartiene alle GUI realizzate con Swing, AWT ed SWT. Infatti un'applicazione scritta con Echo2 è la composizione di oggetti che sono istanza di classi che estendono *nextapp.echo2.app.Component* (es: *Window*, *ContentPane*, *Label*, *Button* etc etc).
In particolare il metodo *init* ritorna la pagina html corrente sotto forma di un oggetto istanza di *Window*. L'oggetto window è stato riempito con un oggetto istanza della classe *ContentPane*, questo oggetto non è altro che un container che occupa tutto lo spazio a sua disposizione. Al suo interno può essere aggiunto un solo Component, quindi di solito si utilizzano quelli (*Row*, *Grid* o *Column*) che ne possono contenere degli altri. Nel nostro caso abbiamo utilizzato un oggetto di tipo *Column*, al quale è possibile aggiungere su ogni riga un altro componente grafico. La parte interattiva della nostra classe è contenuta nella classe anonima aggiunta come *ActionListener* al componente di tipo *Button*:

```
link.addActionListener(...);
```

all'interno del metodo *actionPerformed* viene incrementato un intero e modificata la label inserita in column. Questa porzione di codice viene eseguita sul Server, quindi il client Ajax, precedentemente "disegnato", invia un evento (uno stream XML) di tipo *ActionEvent* al server il quale lo elabora ed invia una risposta (uno stream XML) che modifica l'aspetto dello stesso client. Come tutte le applicazoni Ajax non è necessario ricaricare tutta la pagina HTML, infatti anche nel nostro caso verrà aggiornato solo il Component rappresentato dalla Label label.

FILE BROWSER

Finite le presentazioni iniziamo ad implementare un'applicazione un po' più complessa.
Grazie ad un menu posto sulla sinistra dello schermo si vuole navigare una directory presente sul server, inoltre si vuole fornire un metodo di ricerca che agisca alla pressione di ogni singolo carattere su una casella di testo. Per rendere l'applicazione più completa dal punto di vista funzionale e didattico, sarà presente nel menu la possibilità di visualizzare il file di About che riporta una parte del seguente articolo. Per finire vedremo che l'uso dell'applicazione sarà protetto da una procedura di autenticazione. Come nell'esempio precedente il primo passo sarà quello di implementare una particolare Servlet munita di un metodo *newApplicationInstance* che restituisce un'implementazione concreta dell'interfaccia *ApplicationInstance* che è leggermente diversa da quella dell'esempio precedente.

```
public class Client extends ApplicationInstance {
  ...
  public LoginValidatorIfc validator=null;
  public Switcher switcher=null;
  public Map view=null;
  public String basePath=null;
  public Client(String basePath){
    super();
    this.validator=new SimpleValidator();
    this.basePath=basePath;
    this.view= new HashMap();

  }

  public Window init() {
    this.switcher=new Switcher();
    Window window = new Window();
    ContentPane contentPane = new MainPage();
    window.setContent(contentPane);
     return window;
  }
  public static Client getSessionInstance() {
     return (Client) getActive();
  }
}
```

PROVARE GLI ESEMPI

Per provare gli esempi descritti in questo articolo è sufficiente copiare le directory filebrowser e firstecho2 sotto <tomcat>/webapps. Le due directory sono contenute nella directory prj presente sotto Echo2TestIoP. Una volta copiati i file, basta riavviare tomcat e far puntare un browser ai seguenti url: http://127.0.0.1:8080/filebrowser/app e http://127.0.0.1:8080/firstecho2/app

Il metodo *init* non fa altro che inizializzare un oggetto istanza di *Window* e di inserire al suo interno un contenitore generico di tipo *ContentPane*. Inoltre ha il compito di creare un oggetto di tipo *Switcher* che verrà utilizzato per gestire gli eventi generati dagli oggetti grafici presenti nel menu di sinistra. Nel costruttore invece possiamo notare l'inizializzazione di alcuni oggetti che possono essere utili durante tutta la sessione http. L'oggetto *validator* sarà impiegato per validare le credenziali inserite dall'utente all'atto del *login.*, mentre il *basePath* viene passato dalla Servlet e definisce il path della *webapp* sul filesystem del server. Infine l'HashMap serve a memorizzare i Component che possono essere referenziati da ogni classe del progetto. Il metodo getSessionInstance invece sarà utile per risalire in modo *statico* all'oggetto istanza della classe che stiamo analizzando.

LOGGARSI AL SISTEMA

Per effettuare il login al nostro applicativo è possibile usare sia come username che come password la parola ioprogrammo.

DISEGNIAMO L'APPLICAZIONE

Occupiamoci adesso di disegnare le varie regioni della pagina web che costituisce la nostra applicazione. Questa funzionalità viene implementata nella classe *MainPage*. Il costruttore è molto semplice e non fa altro che invocare il metodo *buildUI*.

```
protected void buildUI(){
  SplitPane mainSplitPane = new SplitPane(
    SplitPane.ORIENTATION_VERTICAL,
    new Extent(80));
  mainSplitPane.setSeparatorWidth(new Extent(1,
                                        Extent.PX));
  add(mainSplitPane);
  mainSplitPane.add(GUIBuilder.getTitleComponent());
  SplitPane bodySplitPane = new SplitPane(
    SplitPane.ORIENTATION_HORIZONTAL,
    new Extent(120));
  bodySplitPane.setSeparatorHeight(new Extent
                                    (1, Extent.PX));
  mainSplitPane.add(bodySplitPane);
  bodySplitPane.setBackground(Client.
                   getSessionInstance().backColor);
  bodySplitPane.add(GUIBuilder.getLeftMenu());
  bodySplitPane.add(GUIBuilder.getCoreContent());
  LoginWindow loginW=new LoginWindow();
  add(loginW);
  Client.getSessionInstance().view.put("loginW",loginW);
  Client.getSessionInstance().view.put("content",this);
 }
```

In questo metodo viene fatto un uso intensivo di oggetti di tipo *SplitPlane*. Questi oggetti dividono lo spazio a loro disposizione in due superfici adiacenti (in orizzontale o in verticale), ognuna delle quali può essere riempita da altri Component. Per prima cosa viene creato l'oggetto *mainSpltiPane* che suddivide il titolo dal resto della pagina. Il titolo, così come gli altri elementi principali, viene creato dal metodo statico della classe *GUIBuilder*. La parte inferiore del precedente SplitPane viene valorizzata da un oggetto dello stesso tipo: *bodySplitPane*. E' evidente che questo contenitore serve a dividere orizzontalmente il menu di sinistra dal body dell'applicazione. Anche queste due regioni della pagina web vengono create dai metodi statici *getLeftMenu* e *getCoreContent* della classe GUIBuilder. A questo punto non ci resta che creare l'oggetto *loginW*, istanza di *LoginWindow*, che definisce una finestra per effettuare il login. Alla fine del metodo buildUI troviamo le seguenti invocazioni

```
Client.getSessionInstance().view.put("loginW",loginW);
Client.getSessionInstance().view.put("content",this);
```

Con ognuna di esse, non facciamo altro che recuperare il Client della sessione corrente, e memorizzare nell'HashMap view gli oggetti grafici che riteniamo possano essere modificati a causa di eventi generati dall'interazione con l'utente.

LOGINWINDOW

Abbiamo già detto che vogliamo fornire la nostra applicazione di un sistema di autenticazione realizzato tramite una form di Login. Per motivi di praticità e vista la natura didattica dell'articolo, non abbiamo ritenuto necessario l'utilizzo di un DB per la memorizzazione delle coppie username-password che identificano gli account degli utenti abilitati al servizio. Infatti la classe *SimpleValidator* memorizza questi dati in una semplice HashMap in cui le *key* rappresentano gli username, mentre gli oggetti memorizzari (i value) determinano la password associata ad ogni username. La classe *SimpleValidator* implementa l'interfaccia *LoginValidatorIfc* che espone oltre ad alcune costanti il solo metodo *validate*. Passiamo adesso ad analizzare le caratteristiche principali della classe *LoginWindow*. Partiamo dalla dichiarazione di classe:

Packages	
nextapp.echo2.app	Provides the core classes and components for creating Echo applications.
nextapp.echo2.app.button	Provides supporting classes for button components including Button, CheckBox, and RadioButton.
nextapp.echo2.app.componentxml	Provides capabilities for Component introspection and loading StyleSheets from XML.
nextapp.echo2.app.componentxml.propertypeer	Provides default peer objects for translating XML property values into Echo property objects.
nextapp.echo2.app.event	Provides core events, listener interfaces, and event management classes.
nextapp.echo2.app.layout	Provides LayoutData implementations which serve to describe the interface between a Component and its parent.
nextapp.echo2.app.list	Provides supporting classes for selection list components including SelectField and ListBox.
nextapp.echo2.app.table	Provides supporting classes for Table components.
nextapp.echo2.app.text	Provides supporting classes for text extry components, such as TextField and TextArea.
nextapp.echo2.app.update	Provides support for synchronizing the state of the server-side application with the client environment.
nextapp.echo2.app.util	Provides utility classes.

Fig. 1: I packages contenuti nel framework

```
public class LoginWindow extends WindowPane
```

Notiamo che la nostra classe estende *WindowPane*, quest'ultima è un particolare contenitore di oggetti che ha la caratteristica di sovrapporsi ad altri Component. Come istanze di classe vengono dichiarati i seguenti oggetti grafici:

```
private TextField usernameField=null;
private PasswordField passwordField=null;
private Label usernameErrorLabel=null;
private Label passwordErrorLabel=null;
```

Il primo rappresenta la casella di testo dove inserire la username, il secondo serve per l'inserimento della password; quest'ultimo nasconde la visualizzazione dei caratteri digitati dall'utente. Gli ultimi due sono oggetti istanze di Label, servono a comunicare all'utente eventuali messaggi di errore in fase di login. Il costruttore della classe implementa il *design* della stessa, ovvero la generazione ed il posizionamento degli oggetti grafici all'interno dello spazio a disposizione. Occupiamoci però delle parti principali, concentrandoci solo su poche righe di codice.

```
public LoginWindow() {
  super();
  this.setModal(true);
  ....
  PushButton button = new PushButton("Login");
  button.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
        processLogin();
     }
   });
   ....
   usernameField = new TextField();
   usernameField.addActionListener(new ActionListener()
{
     public void actionPerformed(ActionEvent e) {
       processLogin();
     }
   });
   ....
   passwordField = new PasswordField();
   passwordField.addActionListener(new ActionListener()
{
     public void actionPerformed(ActionEvent e) {
        processLogin();
     }
   });
       ....
 }
```

Subito dopo aver invocato il costruttore della superclasse, viene utilizzato il metodo *setModal* per rendere modale la finestra che si sta costruendo. In questo modo la finestra in primo piano non può essere nascosta dalla pagina principale della nostra web app.
Un discorso particolare va fatto per l'oggetto *button* istanza di *PushButton*. Questa classe a livello implementativo si comporta a tutti gli effetti come un normale Button, di cui tra l'altro è una sottoclasse;

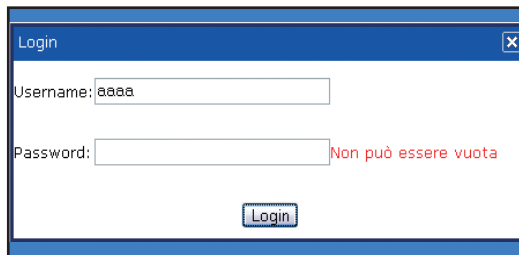



Fig. 3: La form di login di FileBrowser

questa classe però non è contenuta nella librerie standard del Framework. Infatti è fornita dalla libreria *echopointng* che contiene molti widget accattivantche nella maggior parte dei casi colmano i buchi implementativi presenti nella distribuzione standard. In questo caso l'aspetto di un PushButton è molto simile ai pulsanti di Windows e si discosta nettamente dai normali Button molto più simili ad un normale hyperlink HTML.
All'oggetto button e ai due TextField *usernameField* e *passwordField* sono state associate tre classi anonime che implementano ActionListener.
Questo *modus operandi*, come già ribadito in precedenza, è molto simile a quello usato dalle librerie grafiche di J2SE. Tali oggetti delegano la gestione del login al metodo *processLogin*.

```
private void processLogin() {
   String user=this.usernameField.getText();
   String passw=this.passwordField.getText();
   LoginValidatorIfclv=Client.getSessionInstance()
                                              .validator;
   int result=lv.validate(user,passw);
   if(result==LoginValidatorIfc.OK){
       Client.getSessionInstance().logged=true;
       Switcher sw=new Switcher();
       MainPage cp=(MainPage)Client.
             getSessionInstance().view.get("content");
       cp.remove(this);
       Client.getSessionInstance().view.remove("loginW");
        sw.switchAbout();
    }else
if(result==LoginValidatorIfc.ERROR_PASSWORD_EMPTY)
    {
        this.passwordErrorLabel.setText("
                         Non può essere vuota"
       );
    }else
if(result==LoginValidatorIfc.ERROR_USERNAME_EMPTY)
{
```

SNIFFARE I MESSAGGI

Echo2 mette a disposizione dello sviluppatore un metodo molto veloce per monitorare il traffico XML che si scambiano Browser e Server. Basta aggiungere ?debug alla url della nostra applicazione; nel nostro con la url http://127.0.0.1:8080/filebrowser/app?debug vedremo apparire una nuova finestra dove saranno visualizzati i messaggi scambiati.

```
        this.usernameErrorLabel.setText("
                              Non può essere vuota"
        );
    } else
    if(result==LoginValidatorIfc.ERROR_USERNAME){
        this.usernameErrorLabel.setText(" Username
                                          non trovata");
    } else if (result==LoginValidatorIfc.ERROR_GENERAL)
    {
        this.usernameErrorLabel.setText(" Login fallito");
    }
}
```

Una volta recuperati lo username e la password inseriti dall'utente, vengono passati questi dati al metodo *validate* di SimpleValidaor. Il metodo confronta le credenziali dell'utente con quelle presenti sul suo repository, e restituisce un *int* che rappresenta l'esito della validazione. Se l'autenticazione va a buon fine (LoginValidatorIfc.OK) viene rimossa la finestra di login e successivamente viene selezionata la voce *About* dal menu di sinistra. Quest'ultima operazione è realzzata grazie all'utlizzo dell'oggetto istanza di Switcher, che, oltre a selezionare le voci del menu, è un ActionListener delegato alla gestione degli eventi generati dal menu stesso. Nel caso in cui l'autenticazione non va a buon fine, la finestra resta sempre in primo piano ed il sistema evidenzia un messaggio di errore che guida l'utente a capire le cause di tale esito.

L'AUTORE

Roberto Sidoti è Ingegnere Informatico, in passato si è occupato di servizi VoIP; attualmente lavora come Functional Designer per una multinazionale di consulenza specializzata nello sviluppo di componenti per applicazioni Enterprise.

NAVIGARE DA REMOTO

Il servizio principale esposto dalla nosta applicazione è quello di poter navigare e filtrare i file presenti in una directory presente sul server. Ovviamente i file selezionati possono essere scaricati con un semplice link HTML. Per aggiungere un po' di funzionalità tipiche del Web 2.0 decidiamo inoltre di effettuare la selezione dei file in base ai caratteri digitati dall'utente in una casella di testo. L'aggiornamento della lista di file viene effettuato in modo automatico e senza un *submit* esplicito da parte dell'utente. A questo punto non ci resta che approfondire lo studio della classe Switcher già incontrata nelle precedenti classi.

```
public class Switcher implements ActionListener{
  protected Map view=null;
  public Switcher(){
     this.view=Client.getSessionInstance().view;
  }
  ...
}
```

La classe viene utilizzata come ActionListener dei Componenti che prevedono una certa interazione con l'utilizzatore e che sono presenti nel menu di sinistra. Il costruttore non fa altro che valorizzare la Map istanza di classe con quella presente sul Client. Questa classe però gestisce gli input degli utenti all'interno del metodo *actionPerformed*.

```
public void actionPerformed(ActionEvent ev) {
  if(ev.getActionCommand().equals("file")){
     this.switchToFileItem();
  }else if(ev.getActionCommand().equals("about")){
     this.switchAbout();
  }else if(ev.getActionCommand().equals("logOff")){
     this.switchLogOff();
  }else if(ev.getActionCommand().equals("keyStroke")){
     this.switchToFileItem();
  }else if(ev.getActionCommand().equals("docLink")){
     String link=((Button)ev.getSource()).getText();
      this.switchDownload(link);  }
}
```

A questo metodo viene passato un ActionEvent che contiene informazioni utili sull'evento generato da uno dei Component che hanno scelto questa classe come *listener*. Come possiamo notare in base al comando associato all'evento viene invocato uno dei metodi switchXXX presenti nella classe. I primi tre casi gestiscono gli eventi determinati dalla selezione delle rispettive tre voci del menu di sinistra. Il quarto corrisponde alla gestione degli eventi generati dalla tastiera quando il focus è all'interno del TextField associato alla voce *File*. L'ultima istruzione *else if* si occupa di selezionare gli eventi dovuti alla selezione del *link* associato ad uno dei file presenti nella lista filtrata dalla nostra applicazione.

IL FILTRO DEI FILE

Sotto la cartella webapps/filebrowser/files sono presenti alcuni file che possono essere visualizzati in base all'esigenze ed all'interazione con l'utente; il metodo della classe Switcher che gestisce questa funzionalità è switchToFile. Per prima cosa va recuperato l'eventuale testo digitato dall'utente.

```
TextField tf=(TextField)this.view.get("tfFile");
String pat=tf.getText().trim();
```

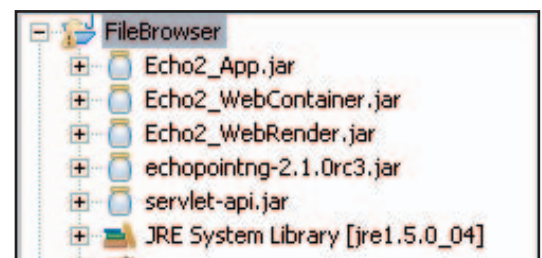



Fig. 4: Il progetto visto da Eclipse

Questo testo serve a determinare un pattern per la ricerca dei file remoti.

```
ArrayList
files=BizLogic.singleton().readFilesWithPattern(pat);
```

La ricerca viene delegata ad un singleton che rappresenta la business logic della nostra applicazione. Rimandiamo per il momento l'analisi del codice di questa classe; supponiamo di aver selezionato i file presenti sulla directory files del server che matchano con il testo da noi digitato, non ci resta che recuperare un riferimento alla zona della pagina che vogliamo riempire e valorizzarlo nel giusto modo.

```
Component body=(Component)this.view.get("body");
GUIBuilder.fillDataGrid(body,files);
```

Per la chiusura del cerchio non ci resta che studiare i caratteri salienti della classe BizLogic

```
public class BizLogic {
  private static  BizLogic sing=null;
  private BizLogic(){}
  public static BizLogic singleton(){
     if(sing==null)
       sing=new BizLogic();
     return sing;
  }
  ...
}
```

Questa prima parte dimostra il modo in cui è stato implementato il pattern *Singleton*, in pratica si tratta di realizzare il costruttore con modificatore *private* e di esporre un metodo statico che restituisce l'istanza di classe dell'oggetto in questione utilizzando eventualmente il costruttore privato. Comunque la vera *logica di business* è contenuta nel metdo *readFilesWithPattern*.

Osservando il suo codice vediamo che per prima cosa viene creato un oggetto istanza di *MyFileFilter*, passandogli la stringa a partire dalla quale si vuole costruire l'espressione regolare necessaria per filtrare i file.

```
FileFilter f2=new MyFileFilter(pat);
```

Successivamente si effettua la scansione e la selezione dei file presenti sotto la directory puntata dall'oggetto *dir*.

```
String basePath=Client.getSessionInstance().basePath;
File dir=new File(basePath+"files/");
ArrayList list=new
ArrayList(Arrays.asList(dir.listFiles(f2)));
```

Una volta recuperata la lista dei file, la stessa viene ordinata sfruttando il metodo statico sort di *java.util.Collections*.

```
Collections.sort(list);
```

L'INTERFACCIA FILEFILTER

La classe *java.io.File* prevede due metodi *list*; il primo è senza argomenti e consente di recuperare tutti i file contenuti in una directory. Il secondo invece prevede il passaggio di un oggetto istanza di una classe che implementa FileFilter. Questa interfaccia prevede il solo metodo *accept* che viene utilizzato per la selezione dei file presenti nella directory e da inserire nella lista di ritorno. Nel nostro esempio abbiamo utilizzato la seguente implementazione:

```
public class MyFileFilter implements
                              FileFilter{
  protected String pattern=null;
  public MyFileFilter(String pattern){
    this.pattern=pattern;
  }
  public boolean accept(File name){
    Pattern p =
Pattern.compile(pattern.toLower
                    Case()+"(.*)");
    Matcher m =
p.matcher(name.getName().toLower
                          Case());
    return m.matches();
  }
}
```

Nel costruttore viene passata la stringa da utilizzare per la selezione dei file. Infatti nel metodo *accept* viene utilizzata a tal proposito un'espressione regolare costruita a partire dalla stringa *pattern*.

Fig. 5: La selezione dei file presenti sulla directory del server

CONCLUSIONI

In questo articolo abbiamo visto come sfruttare Echo2, un framework molto potente per la realizzazione di servizi Web basati sulla tecnologia Ajax. Abbiamo imparato che integrando il framework con la libreria echopointng si possono raggiungere risultati invidiabili che vanno oltre le potenzialità offerte dal framework stesso. Per concludere possiamo affermare che Echo2 è il framework Ajax ideale se non si vuole abbandonare l'approccio classico di programmazione utilizzato nella realizzazione di GUI con J2SE

Roberto Sidoti

REPORT DI QUALITÀ IN ASP.NET

ASP.NET NON OFFRE STRUMENTI PER PRODURRE REPORT PROFESSIONALI SENZA COSTRINGERE L'UTENTE ALL'USO DI INTERNET EXPLORER E ALL'INSTALLAZIONE DI ACTIVEX. SUPERIAMO QUESTO LIMITE CON ACTIVE REPORT, .NET E UN PO' DI INGEGNO

Le applicazioni web somigliano, e sempre più, spesso le sostituiscono, alle applicazioni desktop. In particolar modo questo accade nel mondo delle applicazioni gestionali. Solitamente queste devono essere infarcite di report complessi ed elaborati e non sempre html è in grado di reggere il confronto con l'accattivante estetica di una relazione presentata in Word, corredata di grafici Excel o generata con Crystal Report o altri sistemi di reportistica per le applicazioni desktop. I produttori di componenti di reportistica di stanno sensibilizzando sempre di più a questo problema e così offrono soluzioni per mostrare report di qualità anche nel browser. Purtroppo queste soluzioni, oltre ad essere costose, spesso richiedono l'installazione sulla macchina client di ActiveX o applet Java. Nel caso di ActiveX addirittura si ha la limitazione al solo uso di Internet Explorer e del sistema operativo Windows, limitazione paradossale per un'applicazione web. Inoltre dover installare qualcosa sui client, anche se in modo automatico come accade per gli ActiveX, richiede alcune concessioni da parte del sistemista dell'azienda cliente e questa non è mai una passeggiata.

UNA SOLUZIONE ALTERNATIVA

ActiveReport .NET è tra i pochi prodotti di reportistica nativi per .NET e cioè completamente managed; questa caratteristica offre degli indubbi vantaggi perché semplifica radicalmente il deployment delle applicazioni e non ha il limite di dover interagire con tecnologie non .NET come accade con Crystal Report, ad esempio. A fronte di questi grandi vantaggi, purtroppo non offre un supporto molto sofisticato ad ASP.NET e quindi alla reportistica su web, se non degli scomodi ActiveX che consentono di far apparire nel browser i report in formato nativo oppure in formato PDF, ma sempre passando dall'ActiveX. Questo obbliga ad usare Internet Explorer con gli ActiveX attivi, con tutti gli svantaggi derivanti da questa soluzione: maggiore pericolo per la sicurezza, impossibilità di usare altri browser in alternativa a Internet Explorer. Fortunatamente Active Report .NET offre la possibilità nativa di esportare i report in formato PDF, pertanto possiamo provare a sfruttare a nostro vantaggio questa funzionalità per produrre report da inviare, in formato streaming. In pratica come un flusso di dati binari che attiva Acrobat Reader all'interno del browser dando l'impressione all'utente che il report sia parte della stessa pagina html. Ma andiamo con ordine ed osserviamo un tipico header minimale http che precede una pagina html da inviare ad un browser che ne faccia richiesta:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Content-Type: text/html
Content-Length: 52282
```

Dopo questo preambolo, il browser riceve la pagina html vera e propria e sa che si tratta proprio di html, informazione indicata dall'header Content-Type. È infatti noto che i browser siano capaci di aprire direttamente immagini, documenti Word, documenti PDF e altri formati standard o quanto meno riconosciuti, senza che l'utente debba dare alcune indicazione specifica. Ognuno dei questi formati è associato ad uno specifico MIME Type. Infatti, quando giungono questi formati alternativi all'html, la pagina contiene sempre un header Content-Type specifico che informa il browser su quale tipologia di dati andrà a trattare: a questo punto, se nel browser e, quindi, nel sistema, quel particolare formato è riconosciuto, il browser procederà ad elaborare il documento in modo specifico, diversamente apparirà all'utente la tipica dialog di download per scaricare il flusso binario come file in locale. Ma non è detto che il browser sia in grado di gestire nativamente tutti i formati, questo è vero di solito per html, xml e qualche altro formato grafico. Nella maggior parte dei casi il browser aprirà il programma specifico di gestione di quel formato (ad esempio in presenza di un content/type application/vnd.ms-excel aprirà direttamente

Microsoft Excel all'interno del browser). L'elenco dei MIME Type trattati dal PC si trova nel regestry, alla chiave HKEY_CLASSES_ROOT\MIME\Database \Content Type. È interessante rilevare che la gran parte dei programmi, almeno su sistema operativo Windows, supporta la tecnica di Inplace Activation, che significa che il programma esterno verrà avviato ed eseguito direttamente nella finestra del browser dando all'utente l'impressione che sia direttamente il browser a trattare il file e non un programma esterno. Se il browser non è in grado di gestire direttamente un MIME Type e non vi è alcun programma esterno in grado di trattarlo, semplicemente all'utente verrà proposta la possibilità di eseguire il download del flusso binario trasformando in un file da salvare sul file system.

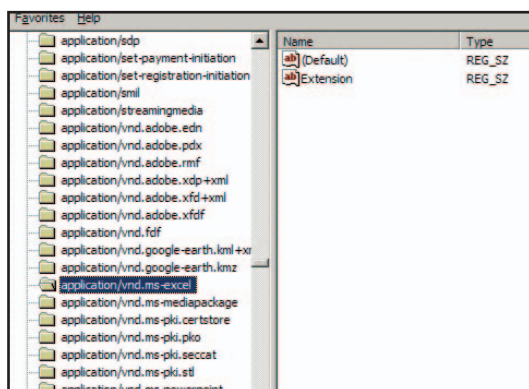



Fig. 1: *La chiave di registry in cui sono classificati i MIME Type*

ACTIVE REPORT.NET IN AZIONE

Dopo aver provveduto a scaricare Active Report .NET e ad installarlo in Visual Studio .NET 2002 – 2003 o 2005, possiamo familiarizzare con l'ambiente grafico di disegno. Dei potenti wizard ci consentiranno di progettare e disegnare il report secondo i consueti canoni di header, footer, bande di dettaglio e subreport che ormai sono lo standard di fatto per tutti i moderni sistemi di reportistica. Per la documentazione specifica e per la documentazione si rimanda direttamente al sito del produttore e ai numerosi esempi che troviamo sulla macchina dopo l'installazione del componente. Allo scopo di velocizzare il nostro esempio, adottiamo direttamente uno dei report di esempio forniti (AnnualReport.csproj) che semplicemente provvederemo a trasformare da applicazione standalone di tipo eseguibile a libreria dll con pochi clic di mouse. Il risultato finale è fornito con i sorgenti allegati.

A questo punto siamo in grado di realizzare la nosta applicazione ASP.NET vera e propria (Report Viewer.csproj) che sarà in grado di generare e

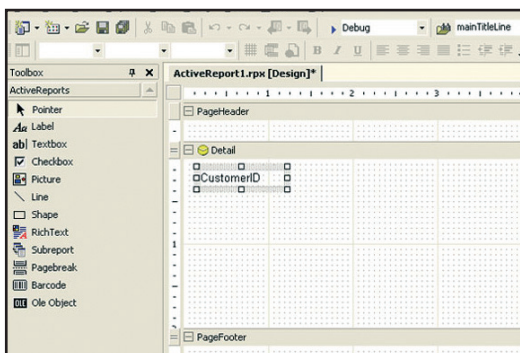



Fig. 2: *Il designer di Active Report .NET direttamente integrato in Visual Studio .NET*

mostrare un report PDF all'interno del browser, sia attivando l'Acrobat Reader nella pagina web stessa che in una finestra separata di tipo popup. Predisponiamo un layout molto semplice per la nostra applicazione web. Nella pagina ASP.NET in questione inseriamo due pulsanti: Mostra Report nell'iFrame e Mostra Report in Finestra dai quali l'utente sarà in grado di visualizzare il report nelle due modalità. Nella pagina è piazzato un ASCX, cioè uno UserWebControl (Sintesi.ascx); ad esso è demandato la visulizzazione del report nell'iFrame. La soluzione è un po' macchinosa ma consente di ridurre quasi a zero l'impatto del report nella propria applicazione evitando di dover scrivere codice specifico di gestione del report in ogni pagina e ad ogni nuovo report, ma con poche semplici istruzioni.

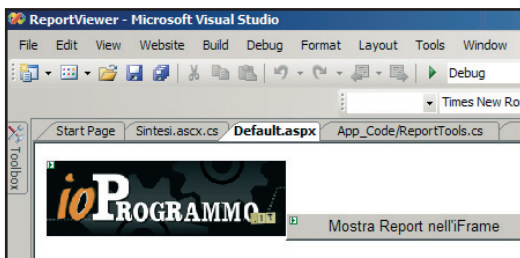



Fig. 3: *Come appare la pagina principale della nostra applicazione nel designer*

IMPLEMENTIAMO LA SOLUZIONE

Innanzitutto realizzeremo una soluzione che permetterà l'aggancio di differenti dll di report in modo dinamico rispetto alla nostra applicazione. In tal modo il nostro sito di esempio diventerà di fatto un visualizzatore universale di report pdf. Prevediamo infatti, nella nostra web.config, una chiave che specificherà il nome dell'assembly del report da mostrare (AnnualReport.dll) e il nome della classe del report da caricare (DataDynamics.ActiveReports.Samples.AnnualReportSample.AnnualReport):

Fig. 4: ***Il report PDF in un iFrame della pagina principale***

```
<appSettings>

        <!--Path dei report -->
    <add key="ReportPath"
     value="D:\Articoli\ioProgrammo\Active Report
            ASP.NET\Disk\ReportViewer\bin\" />

    <!--Elenco dei report -->

    <add key="ioProgrammo.Report.AnnualReport"
 value="AnnualReport.dll,DataDynamics.ActiveReports
  .Samples.AnnualReportSample.AnnualReport" />
</appSettings>
```

Il segreto del nostro sistema è nei metodi statici della classe ReportTools. Qui possiamo trovare il metodo ShowReport, a cui passiamo il reportKey, cioè proprio la chiave che deve avere corrispondenza in quella definita nel web.config, in modo da sapere con esattezza quale assembly di report caricare e quale istanza avviare; gli altri parametri del metodo sono un IDictionary, che conterrà i parametri dal passare al report e l'indicazione di form modale oppure no. Infatti questo metodo viene utilizzato per mostrare il report in una finestra di popup. È interessante notare che i parametri del dictionary verrano trasformati in un querystring da passare alla finestra di popup che conterrà il report e che questa trasformazione viene effettuata in automatico dal metodo GetReport e non dallo sviluppatore. Il popup, in particolare, è proprio la pagina ReportViewer.aspx che contiene esclusivamente il ReportControl.ascx, user control che analizzeremo nello specifico di seguito e che è proprio il cuore del nostro sistema di visualizzazione del report pdf. L'altro metodo, GetReportUrl, viene utilizzato per mostrare il report in un iframe della pagine ospite, cioè della pagina da cui viene effettuata la chiamata. Nell'iframe, infatti, viene semplicemente eseguita la pagina ReportViewer.aspx con il querystring passato per parametro attraverso il dictionary e quindi analogamente a quanto avviene per il popup, ma con la sola differenza che non c'è una finestra esterna ma un iframe.

```
public class ReportTools
{
    public static string GetPageReportUrl
                                    (Page page)
    {
        try
        {
            MethodInfo info =
         page.GetType().GetMethod("GetReportUrl");
            return (string)
              info.Invoke(page, null);
        }
        catch ( Exception ex )
        {
            return null;
        }
    }
    public static bool ShowReport(string
     reportKey, IDictionary parameters, bool modal)
    {
        string value =
    ConfigurationSettings.AppSettings[reportKey];
        string assemblyName =
                        value.Split(',')[0];
        string className =
                        value.Split(',')[1];
        return
      ShowReport(assemblyName, className,
                          parameters, modal);
    }
    public static bool ShowReport
        (string assemblyName, string className,
          IDictionary parameters, bool modal)
    {
        StringBuilder sb = new
                           StringBuilder();
        if ( parameters != null )
        {
            foreach
           ( string key in parameters.Keys )
            {
            sb.Append(key);
            sb.Append("=");
        sb.Append(parameters[key]);
            sb.Append("&");
            }
        }
        if ( modal )
    HttpContext.Current.Response.Write("
            <BODY onLoad=\"javascript:window.
  showModalDialog('../ReportViewer.aspx?Assembly="
 + assemblyName + "&Type=" + className + "&" +
    sb.ToString() + "', 'Link','dialogToolbar=0;
dialogLocation=0;help=0;status=0;dialogMenubar=0;
scroll=0;resizable=0;left=250;top=250;dialogWidth:
            630px;dialogHeight:310px');\"");
        else
    HttpContext.Current.Response.
```

Vito Vessia progetta e sviluppa applicazioni e framework in .NET, COM(+) e Delphi occupandosi degli aspetti architetturali. Scrive da anni per le preincipali riviste italiane di programmazione ed è autore del libro "Programmare il cellulare", Hoepli, 2002, sulla programmazione dei telefoni cellulari connessi al PC con protocollo standard AT+. Può essere contattato tramite e-mail all'indirizzo
vvessia@katamail.com

```
Write("<SCRIPT>javascript:window.open('../ReportVi
        ewer/ReportViewer.aspx?Assembly=" +
    assemblyName + "&Type=" + className + "&"
        + sb.ToString().Replace("'", "~") + "','Link',
    'toolbar=0,location=0,directories=0,status=0,
  menubar=0,scrollbars=1,resizable=1,width=950,
        height=600,left=30,top=80');</SCRIPT>");
            return true;
    }
    public static string GetReportUrl(string
                reportKey, IDictionary parameters)
    {
            string value =
    ConfigurationSettings.AppSettings[reportKey];
            string assemblyName =
                            value.Split(',')[0];
            string className =
                            value.Split(',')[1];
            return
        GetReportUrl(assemblyName, className,
                                parameters);
    }
    public static string GetReportUrl(string
                assemblyName, string className,
                        IDictionary parameters)
    {
            StringBuilder sb =
                        new StringBuilder();
            if ( parameters != null )
            {
                    foreach
                ( string key in parameters.Keys )
                    {
                    sb.Append(key);
                    sb.Append("=");
            sb.Append(parameters[key]);
                    sb.Append("&");
                    }
            }
            return "../ReportViewer/
    ReportViewer.aspx?Assembly=" + assemblyName +
    "&Type=" + className + "&" + sb.ToString();
    }
}
```

Diventa così banale invocare la visualizzazione di un report in un popup o in un iframe dalla nostra pagina web, ed infatti osserviamo il codebehind completo della nostra pagina default.aspx:

```
private bool _showInternalReport;
public string GetReportUrl()
{
    if ( _showInternalReport )
    {
            Hashtable cfg = new Hashtable();
            cfg["ID"] = "chiave 1";
            return
```

```
        ReportTools.GetReportUrl("ioProgrammo.Report.
                            AnnualReport", cfg);
    }
    else
            return null;
}
private void cmdShowInternalReport_Click
                (object sender, System.EventArgs e)
{
    _showInternalReport = true;
}
private void cmdShowWindowedReport_Click
                (object sender, System.EventArgs e)
{
    Hashtable cfg = new Hashtable();
    cfg["ID"] = "chiave 1";
ReportTools.ShowReport("ioProgrammo.Report.
                    AnnualReport", cfg, false);
}
public static string GetPageReportUrl(Page page)
{
    try
    {
            MethodInfo info =
        page.GetType().GetMethod("GetReportUrl");
            return (string) info.Invoke(page,
null);
    }
    catch ( Exception ex )
    {
            return null;
    }
}
public string GetReportUrl()
{
    if ( _showInternalReport )
    {
            Hashtable cfg = new Hashtable();
            cfg["ID"] = "chiave 1";
            return
        ReportTools.GetReportUrl("ioProgrammo.
                Report.AnnualReport", cfg);
    }
    else
            return null;
}
```

Si è potuto certamente osservare che i due eventi click dei due bottoni che fanno attivare il report, contengono solo una manciata di istruzioni che fanno riferimento ai metodi di ReportTools appena introdotti.

IL MOTORE DEL SISTEMA

A questo punto non ci resta che concentrarsi sul motore del sistema e cioè sullo usercontrol

Datadynamics
http://www.datadynamics.com

RFC 2616: Hypertext Transfer Protocol 1.1
http://www.w3.org/Protocols/rfc2616/rfc2616.html

ReportControl.ascx e, in particolare, sul suo code-behind. Osserviamolo:

```
using DataDynamics.ActiveReports;
using DataDynamics.ActiveReports.Export.Pdf;
private void Page_Load(object sender,
System.EventArgs e)
{
        // Put user code to initialize the page here
        //string reportFile = ConfigurationSettings.
                AppSettings["ReportTempPath"] +
        //
          Path.GetFileName(Path.GetTempFileName());
        string reportFile = Path.GetTempFileName();
        Hashtable pars = new Hashtable();
        string assemblyName = null;
        string typeName = null;
        foreach ( string key in
Request.QueryString.AllKeys )
        {
                if ( key == null ) continue;
                switch ( key.ToUpper() )
                {
                        case "ASSEMBLY":
                        assemblyName =
        ConfigurationSettings.AppSettings["ReportPath"]
                        + Request.QueryString[key];
                                break;
                        case "TYPE":
        typeName = Request.QueryString[key];
                                break;
                        default:
                                pars[key] =
           Request.QueryString[key].Replace("~", "");
                                break;
                }
        }
        ActiveReport report = (ActiveReport)
    AppDomain.CurrentDomain.CreateInstanceFromAndU
                nwrap( assemblyName, typeName, true,
                        BindingFlags.DeclaredOnly |
                BindingFlags.Public |
                        BindingFlags.NonPublic |
                BindingFlags.Instance |
      BindingFlags.GetField ,  null, new object[] {pars},
                                null, null, null );
        report.Run(true);

        PdfExport pdf = new PdfExport();
        //XlsExport pdf = new XlsExport();
        Response.ClearContent();
        Response.ClearHeaders();
        Response.ContentType = "application/pdf";
        //Response.ContentType =
                        "application/vnd.ms-excel";
        pdf.Export(report.Document, reportFile);
        Response.WriteFile( reportFile );
        Response.Flush();
        Response.Close();
        report.Document.Dispose();
        report.Dispose();
                File.Delete(reportFile);
}
```

Si deve considerare che questo ascx è ospitato in ReportViewer.aspx che appare, quindi, nel popup o nell'iframe, ma sempre perfettamente configurato con un querystring contentente sia le indicazioni del report da caricare, che i parametri da passare ad esso. In particolare dal querystring vengono estratti i parametri ASSEMBLY e TYPE rispettivamente indicanti il nome dell'assembly contenente il report ActiveReport da mostrare il full qualified name della classe da istanziare. Tutti gli altri parametri del querystring vengono semplicemente riassemblati in un nuovo dictionary che verrà passato al costruttore della classe report specificata, al momento della sua istanziazione. A questo punto, usando la reflection ed in particolare il metodo CreateInstance FromAndUnwrap di AppDomain si provvede a caricare in memoria l'assembly del report e ad istanziarne la classe passandogli il dictionary con i parametri indicati dal chiamante e riassemblati nel nuovo dictionary. Il report viene a questo punto processato dal motore di ActiveReport (il metodo Run) ed esportato al volo in un file temporaneo (Path.GetTemp FileName) in formato pdf, utilizzando la libreria di esportazioni in pdf di ActiveReport. E qui arriva finalmente la fase più interessante: la trasmisisone dello stream binario pdf al browser. La cosa richiede una preparazione e cioè l'impostazione del content type della pagina con il riferimento al pdf (application/pdf), come descritto in apertura di articolo, viene svuotato l'elenco degli header http eventualmente già in cache in modo che il flusso da trasmettere non sia contaminato da impurità e viene semplicemente invocata una Response.WriteFile a cui viene passato il file pdf appena processato ed esportato. A questo punto si invoca una Flush della pagina in modo da completarne la trasmissione del flusso, viene chiusa la comunicazione e cancellato il pdf temporaneo ormai inutile. Il gioco è fatto perché il nostro browser, come per magia, riconoscerà il formato pdf e attiverà semplicemente il Reader nella pagina riassemblando il flusso binario ricevuto che altro non è che il nostro report pdf!

CONCLUSIONI

L'utilizzo di questa tecnica consente di evitare l'installazione si scomodi ActiveX o applet Java e rende il report PDF compatibile con tutti i sistemi e con tutti i browser

Vito Vessia

REGEX E VB.NET BINOMIO VINCENTE

IL MICROSOFT .NET FRAMEWORK È DOTATO DI UN MOTORE DI ESPRESSIONI REGOLARI MOLTO POTENTE, ACCESSIBILE DA QUALSIASI LINGUAGGIO .NET, INCLUSO IL NOSTRO VISUAL BASIC. IMPARIAMO COME SFRUTTARLO AL MEGLIO

Le Espressioni Regolari nascono con alcuni linguaggi di programmazione come Perl e Awk e sono poco note ai programmatori Microsoft Windows. Rappresentano tuttavia un metodo potente e flessibile per la manipolazione di testi. Se non si ha idea di cosa siano le espressioni regolari, basti pensare ai caratteri jolly che si utilizzano al prompt dei comandi per indicare un gruppo di file (come in *.txt) oppure ai caratteri speciali che si possono utilizzare con l'operatore *Like* nelle query SQL. Le espressioni regolari si utilizzano principalmente per trovare corrispondenze di modelli (pattern) su stringhe, e costituiscono uno strumento tanto difficile ed ostico (soprattutto all'inizio) quanto potente ed utile.
Una delle applicazioni più interessanti, è legata al riconoscimento di linguaggi che fanno un uso pesante di markup, come i documenti XML e le pagine HTML. Grazie alla flessibilità delle espressioni regolari, si può analizzare il formato di un intero file di testo, così come si può verificare che una semplice stringa soddisfi un certo formato, ad esempio una data o un indirizzo e-mail

IL LINGUAGGIO

Le espressioni regolari possono essere considerate come un linguaggio di programmazione altamente specializzato ed ottimizzato per la modifica del testo, e come ogni linguaggio, richiede del tempo per essere digerito in tutte le sue particolarità. Il linguaggio utilizza due tipi fondamentali di caratteri: caratteri effettivi (normali) e metacaratteri.
Probabilmente, i meno giovani di noi, ricorderanno i metacaratteri *?* e *, utilizzati con il file system Dos che rappresentano un carattere singolo (?) ed un gruppo di caratteri (*).
Ad esempio il comando Dos: *Copy *.txt A*: permette di copiare tutti i file con estensione txt sul disco nell'unità A. Il metacarattere * rappresenta qualsiasi nome di file davanti all'estensione txt.
Le espressioni regolari estendono notevolmente questo concetto di base, poiché forniscono un ampio set di metacaratteri così da rendere possibile descrivere espressioni molto complesse per criteri di ricerca o di modifica del testo.
Tra i metacaratteri più usati trovano ampio spazio il punto (.), che indica un qualsiasi carattere singolo, l'asterisco (*), che indica la presenza di zero o più occorrenze del carattere precedente, il simbolo +, che indica la presenza di una o più occorrenze del carattere precedente, il punto interrogativo (?), che indica la presenza di zero o una occorrenza del carattere precedente.
Per chiarirci le idee possiamo fare un piccolissimo esempio. Pensiamo a quale potrebbe essere la funzionalità desiderata da un TextBox destinato a contenere un importo in Euro, come possiamo controllare ciò che ha digitato l'utente? Se il contenuto del campo deve essere inserito, ad esempio, in un campo di database di tipo *Decimal*(10,2), il modello che lo descrive può essere:

```
^[0-9]{1,10}\.[0-9]{2}$
```

Capiremo più avanti cosa vuol dire

ELEMENTI DEL LINGUAGGIO

L'insieme dei metacaratteri che definiscono il linguaggio delle espressioni regolari, viene raggruppato nelle seguenti categorie

- Caratteri di escape
- Classi di caratteri
- Sostituzioni
- Asserzioni atomiche di ampiezza zero
- Quantificatori

- Costruttori di raggruppamento
- Costrutti di riferimento
- Costrutti di alternanza
- Costrutti vari

I *caratteri di escape* vengono utilizzati per confrontare singoli caratteri. Si possono utilizzare per trattare i caratteri non stampabili (come i caratteri *BackSpace* e *tab*) oppure per segnalare i caratteri che assumono un significato speciale all'interno dei pattern dell'espressione regolare.

Sono caratteri di escape:

\b Rappresenta il carattere *BackSpace*, ma solo se utilizzato tra parentesi quadre, altrimenti rappresenta un limite di parola
\t Rappresenta il carattere di Tabulazione
\r Rappresenta il carattere di ritorno carrello
\n Rappresenta il carattere a capo
***** Quando il backslash (\) è seguito da un qualsiasi carattere che non è compreso tra i caratteri di escape, allora rappresenta il carattere stesso

Una *classe di caratteri* permette di confrontare un carattere con l'insieme di caratteri racchiuso tra parentesi quadre, come [abcd]. Quando si usa questa sintassi, e si devono includere dei caratteri speciali, non è necessario farli precedere da una sequenza di escape. Gli unici due caratteri che assumono un significato speciale all'interno delle parentesi quadre e che, quindi, non obbediscono a questa regola, sono il trattino "-" e la parentesi quadra chiusa.

[abc] Quasiasi carattere compreso tra parentesi quadre
[^abc] Quasiasi carattere escluso quelli tra parentesi quadre
[a-zA-Z] Il carattere – (trattino) consente di indicare un intervallo di caratteri, in questo caso l'espressione di esempio rappresenta qualsiasi carattere minuscolo o maiuscolo, ma non rappresenta le lettere accentate
\w Rappresenta un qualsiasi carattere alfanumerico comprese le lettere accentate
\d Rappresenta una cifra decimale, analogo a [0-9]

Le *Sostituzioni*, unitamente ai caratteri di escape, sono i soli costrutti speciali che possono essere utilizzati all'interno dei pattern di sostituzione, in particolare, le *sostituzioni* possono essere impiegate soltanto all'interno dei criteri di sostituzione.

${nomeGruppo} Sostituisce l'ultima sottostringa rappresentata da un gruppo dal nome racchiuso tra parentesi.
$_ Sostituisce l'intera stringa sorgente

Le *asserzioni atomiche di ampiezza zero* indicano la posizione dove dovrebbe essere la stringa di confronto, ma non consumano caratteri. Ad esempio, $ indica la fine della stringa, pertanto l'espressione regolare *edmaster$* corrisponde a qualsiasi parola *edmaster* che si trovi subito prima della fine di una riga, senza comprendere anche il carattere di fine riga.

^ Indica l'inizio della stringa
$ Indica la fine della stringa
\b Indica il primo e l'ultimo carattere di una parola delimitata da spazi o da altri simboli di punteggiatura

I *quantificatori* indicano che una sottostringa deve essere ripetuta per un certo numero di volte. Un quantificatore può essere applicato al carattere, al gruppo o alla classe di caratteri immediatamente precedente ad esso. Ad esempio, \b[aeiou]+\b corrisponde a tutte le parole costituite solo da vocali. Le espressioni regolari di .NET Framework supportano due categorie di quantificatori: *greedy* e *lazy*. Un quantificatore greedy, confronta sempre il maggior numero di caratteri possibile, un quantificatore lazy, invece, tenta di confrontare il minor numero di caratteri possibile.

***** Indica zero o più corrispondenze
+ Indica una o più corrispondenze
? Indica zero o una corrispondenza

I *costruttori di raggruppamento* permettono di acquisire e denominare gruppi di sottoespressioni e di accrescere l'efficienza delle espressioni regolari. Ad esempio, (edmaster)+ rappresenta sequenze ripetute della stringa "*edmaster*".
Sono *costrutti di raggruppamento*:

(substr) acquisisce la sottostringa rappresentata, le acquisizioni sono numerate automatica-

NOTA

RIFERIMENTI

In letteratura esistono numerosi esempi di espressioni regolari, si possono trovare nella finestra di dialogo Regular expression Editor di Visual Studio oppure in uno dei numerosi siti ad esse dedicati, tra gli altri www.regexlib.com

NOTA

MODULI DI RICHIESTA/ RISPOSTA

La classe HttpWebRequest fornisce supporto per le proprietà e i metodi definiti in WebRequest e per le proprietà e i metodi aggiuntivi che consentono all'utente di interagire direttamente con i server tramite HTTP. La classe WebRequest rappresenta la classe base astratta per il modello di richiesta/risposta necessario per l'accesso ai dati da Internet.

	mente
(?<name>)	acquisisce la sottoespressione corrispondente in un nome di gruppo o un nome di numero
(?:)	Gruppo di non acquisizione

I *costrutti di riferimento all'indietro* (backreference) consentono di referenziare un precedente gruppo di caratteri del pattern dell'espressione regolare attraverso il numero o il nome del gruppo.

\ number	Riferimento all'indietro ad un gruppo precedente. Ad esempio, (\w)\1 trova caratteri alfanumerici doppi.
\k<name>	Riferimento all'indietro denominato. È possibile utilizzare virgolette singole al posto delle parentesi tonde, ad esempio \k'char'.

I *costrutti di alternanza* forniscono un modo per specificare alternative o/o

|	Corrisponde ad uno qualsiasi dei termini separati dal carattere | (barra verticale), ad esempio qui|quo|qua. La corrispondenza più a sinistra prevale.

I *costrutti vari* comprendono costrutti che consentono di modificare una o più opzioni delle espressioni regolari.

(? imnsx - imnsx)	Abilita o disabilita le opzioni dell'espressione regolare. Ad esempio permette la distinzione tra maiuscole e minuscole, affinché possano essere attivate o disattivate nel bel mezzo di un pattern.
(?#)	da utilizzare quando si vogliono inserire all'interno di un'espressione regolare dei commenti inline

LE CLASSI

Non ci rimane che capire come integrare il linguaggio alla base delle espressioni regolari con Visual Basic. Tutte le classi che forniscono l'accesso alla gestione delle espressioni regolari di VB.Net sono racchiuse nel namespace: *System.Text.Regular Expressions*. Lo stesso namespace può essere utilizzato da qualsiasi piattaforma o linguaggio eseguito in Microsoft .NET Framework.

La classe *Regex* è la classe più importante di questo gruppo, e qualsiasi codice relativo alle espressioni regolari istanzia almeno un oggetto di questa classe o utilizza uno dei suoi metodi statici. Un oggetto *Regex* rappresenta una espressione regolare immutabile. I metodi statici di *Regex*, permettono di utilizzare le altre classi messe a disposizione per la gestione delle espressioni regolari senza crearne in modo esplicito le istanze. In pratica, utilizzare un metodo statico equivale a costruire un oggetto *Regex*, utilizzarlo una sola volta e distruggerlo immediatamente.

L'oggetto *Regex* si istanzia passandogli il pattern di ricerca, scritto utilizzando il linguaggio delle espressioni regolari. Per valutare un campo valuta, come nell'esempio precedente, possiamo scrivere:

```
Dim re As New Regex("^[0-9]{1,10}\.[0-9]{2}$")
```

La classe *Match* rappresenta una singola corrispondenza, risultato dell'analisi basata su un'espressione regolare.

Un oggetto di tipo *Match*, si può ottenere utilizzando il metodo *Match* della classe *Regex* e conterrà la prima corrispondenza nella stringa di input. L'oggetto *Match* non ha alcun costruttore pubblico, pertanto si può ottenere soltanto utilizzando il metodo *Match* della classe *Regex* oppure iterando sulla collezione *MatchCollection*.

La classe collezione *MatchCollection* rappresenta una collezione di corrispondenze individuate e non sovrapposte. *MatchCollection* non dispone di costruttori pubblici ed è una collezione a sola lettura. Si può creare un oggetto di tipo *MatchCollection* solo utilizzando il metodo *Regex.Matches*.

Le principali proprietà della classe *Match* sono:

Value	restituisce la sottostringa confrontata
Length	restituisce la lunghezza della sottostringa confrontata
Index	restituisce la posizione in cui è stato trovato il primo carattere della sottostringa acquisita, nella stringa originale.
Success	restituisce *True* se la ricerca di corrispondenze ha avuto esito positivo, *False* nel caso contrario

GLI IMPORT DA UTILIZZARE

Per evitare di scrivere righe di codice troppo lunghe, possiamo assumere di aver scritto le seguenti istruzioni di Imports: Imports System.IO Imports System.Text Imports System.Text.RegularExpressions Imports System.Net

ESPLORIAMO IL WEB

Mettiamo a frutto le nostre conoscenze e realizziamo un'applicazione che, dato un indirizzo internet, esplora il contenuto della pagina ed individua tutti i link e le immagini presenti nella pagina stessa.
Disegniamo l'interfaccia e posizioniamo i seguenti controlli:

- Due *ListView* dai nomi *lvwImmagini* e *lvwLinks*, deputati a visualizzare i risultati dell'esplorazione
- Un *TextBox* dal nome *txtURL* in cui l'utente dovrà scrivere l'indirizzo del sito da esplorare
- Due *OptionButton* dai nomi *OptImmagini* e *OptLinks* necessari per selezionare il tipo di informazione richiesta
- Un *Button* dal nome *ButtonEsplora* per avviare il processo di esplorazione e verifica

Dichiariamo le variabili che utilizzeremo nella nostra applicazione:

```
    Private UrlSelezionato As String
    Private Immagini As
System.Collections.Generic.Dictionary(Of String,
ImageAttributes)
    Private RichiestaHttp As HttpWebRequest
    Private RispostaHttp As HttpWebResponse
    Private TestoPaginaWeb As String
    Private regExPattern As String
```

Scriviamo il codice necessario al corretto funzionamento dell'applicazione nell'evento *click* di *ButtonEsplora*

```
Private Sub ButtonEsplora_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs)
Handles ButtonEsplora.Click
End Sub
```

Il primo controllo che dobbiamo fare è sul corretto inserimento dell'URL da parte dell'utente. L'URL deve iniziare con la stringa *http://* in caso contrario avvisiamo l'utente della scrittura errata dell'indirizzo Web, ed usciamo dalla procedura

```
        If
Microsoft.VisualBasic.Left(Trim(txtURL.Text), 7) <>
"http://" Then
            MsgBox("L'indirizzo deve iniziare con
http://", MsgBoxStyle.Critical, "Attenzione")
            Exit Sub
        End If
```

Eliminato questo primo inconveniente, possiamo procedere con la lettura del flusso di dati proveniente dal sito il cui indirizzo è stato inserito nel TextBox: *txtURL*.

```
        ' Otteniamo una stringa contenente
        'il codice sorgente della pagina Web.
        Try
            TestoPaginaWeb =
ConvertiFlussoInTesto(GetFlussoHttp (txtURL.Text))
        Catch exp As Exception
            MsgBox(exp.Message,
                MsgBoxStyle.Critical, "Attenzione")
            Exit Sub
        End Try
```

Tralasciamo, il funzionamento delle due procedure *ConvertiFlussoInTesto* e *GetFlussoHttp*, ci basti sapere che, a questo punto, la variabile *TestoPaginaWeb* contiene il testo della pagina Web selezionata. Se l'utente ha selezionato l'OptionButton dei Links, allora scriviamo un classico *If* in cui puliamo la corrispondente ListView (lvwLinks) ed impostiamo l'espressione regolare che ci permette di trovare tutti i link sparsi per la pagina. Per analizzare un link HTML come ad esempio:

```
 <a href="http://www.facileprivacy.it">Tutto sulla
Privacy</a>
```

Dobbiamo utilizzare i seguenti costrutti

<a	Individua l'inizio del link HTML
\s+	Indica che ci possono essere uno o più spazi vuoti

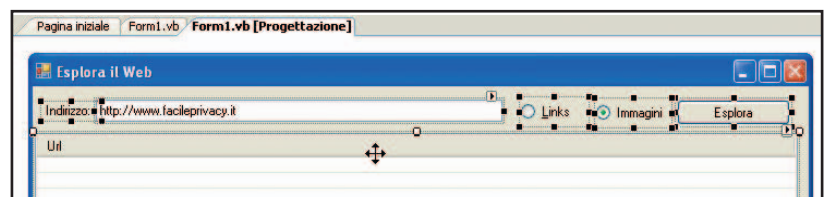



Fig. 1: La disposizione dei controlli nella form

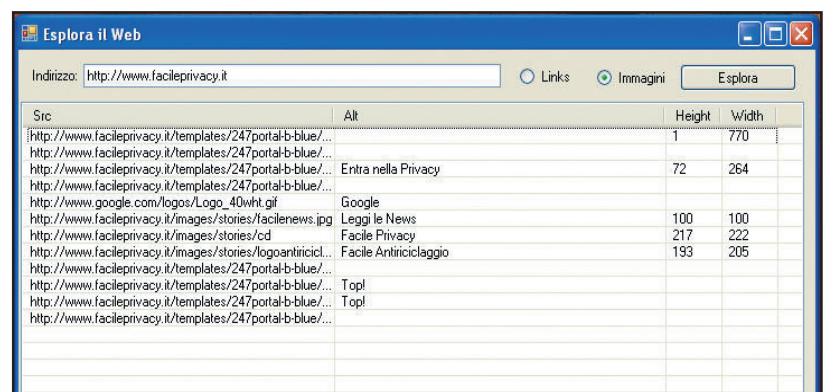



Fig. 2: L'analisi dei link contenuti nella pagina

NOTA

LE COLLEZIONI

La classe collezione GroupCollection rappresenta un insieme di gruppi acquisiti e lo restituisce in una singola corrispondenza. La collezione non è modificabile (sola lettura) e non dispone di costruttori pubblici. L'unico modo per istanziare un oggetto di GroupCollection è costituito dall'utilizzo della proprietà Match.Groups.

UN METODO PER ESTRARRE LE IMMAGINI

```
Private Sub
MostraImmaginiInListView()
        For Each source As String In
Immagini.Keys
            'Crea un oggetto
ListViewItem e impostare il testo
della prima colonna.
            Dim lvi As New
ListViewItem()
            lvi.Text = source
            ' Impostare il testo nelle
colonne rimanenti e aggiungere
l'oggetto ListViewItem al controllo
            ' ListView.
            Dim imgAttr As
AttributiImmagine =
CType(Immagini(source),
AttributiImmagine)
            With lvi.SubItems
                .Add(imgAttr.Alt)
                .Add(imgAttr.Height)
                .Add(imgAttr.Width)
            End With
            ' Aggiunge l'oggetto
ListViewItem all'insieme di
elementi del controllo ListView.

lvwImmagini.Items.Add(lvi)
        Next
End Sub
```

href	Individua la continuazione della sintassi di un link HTML
\s*	Indica che ci possono essere zero o più spazi vuoti
=	Individua la continuazione della sintassi di un link HTML
\s*	Indica che ci possono essere zero o più spazi vuoti
""?	Indica che ci possono essere Zero o una virgoletta (in sequenza escape)
(	Indica che inizia il gruppo che definisce una sottostringa: l'URL del link.
[^"" >]+	Indica una o più corrispondenze di qualsiasi carattere eccetto quelli tra parentesi.
)	Indica la fine del primo gruppo che definisce una sottostringa
""?	Indica che ci possono essere Zero o una virgoletta (in sequenza escape)
>	Individua la continuazione della sintassi di un link HTML
(.+)	Indica un gruppo corrispondente a qualsiasi carattere: il testo del link
****	Individua la fine della sintassi di un link HTML

Possiamo quindi scrivere:

```
        If optLinks.Checked Then
            MostraStato("Connessione alla
                    pagina web in corso....  ")
            lvwLinks.Items.Clear()
            regExPattern =
"<a\s+href\s*=\s*""?([^"" >]+)""?>(.+)</a>"
        Else
```

Nel ramo *Else* puliamo lvwImmagini ed impostiamo l'espressione regolare che ci permette di recuperare tutte le immagini sparse per la pagina.

L'espressione regolare per ottenere le immagini contenute in una pagina web è concettualmente simile a quella vista in precedenza per i link, ma risulta più complessa poiché vengono acquisiti fino a quattro diversi attributi, che possono apparire in qualsiasi ordine.
Per evitare di scrivere una espressione regolare molto complessa, dobbiamo porre alcune condizioni, affinché i tag HTML <img> possano venire elaborati.

- L'attributo src è sempre presente ed è l'unico attributo sempre necessario.
- L'attributo src precede gli attributi width ed height. In caso contrario, non è possibile ottenere width ed height.
- L'attributo alt segue gli attributi width ed height. In caso contrario, non è possibile ottenere alt.
- Gli attributi width ed height possono seguire src in qualsiasi ordine reciproco.

Questi presupposti ci permettono di recuperare tutte le immagini contenute nella pagina web, l'unico inconveniente è che alcuni dei dati sugli attributi potrebbero non essere rilevati.

Alcuni costrutti che dovremo utilizzare sono:

[^>]+	Corrisponde a qualsiasi carattere scluso il carattere >. In questo modo possiamo spostarci ai successivi attributi.
(?:	Avvia un gruppo di non acquisizione. In questo modo possiamo rendere facoltativi tutti gli attributi escluso l'attributo *src*.
|	Costrutto di alternanza utilizzato con gli attributi *width* ed *height* come espressione Or.

```
Immagini = New System.Collections.Generic.
Dictionary(Of String, AttributiImmagine)
lvwImmagini.Items.Clear()
regExPattern = "<img[^>]+(src)\s*=\s*""?([^
"">]+)""?(?:[^>]+(width|height)\s*=\s*""?([^
"">]+)""?\s+(height|width)\s*=\s*""?([^
"">]+)""?)?(?:[^>]+(alt)\s*=\s*""?([^"">]+)""?)?
"
End If
```

Instanziamo l'oggetto *Regex* con il pattern definito in precedenza, e l'opzione *IgnoreCase* in modo da evitare di fare distinzione tra lettere

maiuscole e minuscole.

```
Dim re As New Regex(regExPattern,
                    RegexOptions.IgnoreCase)
```

Instanziamo l'oggetto *Match* utilizzando il metodo *Match()* dell'oggetto *Regex* che restituisce una sola corrispondenza per volta.

```
Dim m As Match = re.Match(TestoPaginaWeb)
```

Poiché il metodo *Match()* dell'oggetto *Regex* restituisce una sola corrispondenza per volta, dobbiamo utilizzare la proprietà *Success* per testare se è presente un'altra corrispondenza, e la proprietà *NextMatch()* che passa alla successiva corrispondenza provocando l'iterazione.
Nel caso di un'immagine (*optImmagini.Checked=True*) possiamo utilizzare la proprietà *Match.Groups* che ottiene l'insieme di gruppi che corrispondono all'espressione regolare.
Poiché il pattern delle espressioni regolari ottiene alcune caratteristiche di ordinamento per gli attributi *width* ed *height*, allora dobbiamo determinare l'attributo elencato per primo nell'ordine, in modo da assegnare il valore dell'elemento *Group* corretto.
A questo punto possiamo chiamare la procedura *AddImage* per aggiungere un'istanza dell'oggetto *AttributiImmagine* ad una classe *Hashtable* che ci permetterà di gestire l'elenco delle immagini presenti nella pagina Web.
Nel caso di un link (*optImmagini.Checked =False*) utilizziamo sempre la proprietà *Match.Groups* e mostriamo i risultati nella ListView.
Il codice da scrivere è:

```
While m.Success
    If optImmagini.Checked Then
        Dim width As String
        Dim height As String
        If m.Groups(3).Value.ToLower =
                                "width" Then
            width = m.Groups(4).Value
            height = m.Groups(6).Value
        Else
                ' L'attributo height è stato
                'immesso per primo
            width = m.Groups(6).Value
            height = m.Groups(4).Value
        End If
        AddImage(New
        AttributiImmagine(m.Groups(2).Value, _
            m.Groups(8).Value, height, width))
    Else
        Dim lvi As New ListViewItem()
        lvi.Text = m.Groups(1).Value
            lvwLinks.Items.Add(lvi)
        End If
        ' Continua il ciclo fino
        'alla corrispondenza
        'successiva.
        m = m.NextMatch()
End While
```

Infine, nel caso delle immagini, non ci resta che mostrarne i risultati ottenuti nella ListView corrispondente. Allo scopo utilizziamo la procedura *MostraImmaginiInListView* che deve semplicemente scorrere una classe *HashTable* ed aggiungere e mostrare nella ListView ognuno degli oggetti *attributiImmagine* presenti nella classe *HashTable*.
Nel caso dei link, i risultati sono già stati mostrati in precedenza per cui dobbiamo soltanto mostrare un messaggio di avviso nel caso in cui non è stato trovato alcun link

```
If optImmagini.Checked Then
    If Immagini IsNot Nothing Then
        MostraImmaginiInListView()
    Else
        MsgBox("Nessuna Immagine trovata ",
    MsgBoxStyle.Information, Me.Text)
    End If
Else
    If lvwLinks.Items.Count = 0 Then
        MsgBox("Nessun Link Trovato",
    MsgBoxStyle.Information, Me.Text)
    End If
End If
```

Luigi Buono

NOTA

I GRUPPI

La classe Group rappresenta i risultati di un singolo gruppo di acquisizione. Grazie ai quantificatori, in una singola corrispondenza Group si può acquisire una o più stringhe o non acquisirne alcuna. Le istanze di Group vengono restituite dalla proprietà Match.Groups(numgruppo) o Match.Groups("nomegruppo") se si utilizza il costrutto di raggruppamento "(?<nomegruppo>)".

DAL FLUSSO AL TESTO

```
Function
ConvertiFlussoInTesto(ByVal
stmSource As Stream) As String
        Dim sr As StreamReader =
Nothing

        If Not IsNothing(stmSource)
Then
            Try
                sr = New
StreamReader(stmSource)
                ' Legge e restituisce
l'intero contenuto del flusso.
                Return sr.ReadToEnd
            Catch exp As Exception
                ' Inoltra il messaggio
di errore personalizzato da
GetFlussoHttp().
                Throw New
Exception()
            Finally
                ' Pulisce sia Stream
che StreamReader.
                RispostaHttp.Close()
                sr.Close()
            End Try
        End If
        Return Nothing
    End Function
```

ASP.NET E JAVASCRIPT DUE MONDI VICINI

PER SUA NATURA ASP.NET TENDE A NASCONDERE AL PROGRAMMATORE LE PARTI DI SCRIPTING CHE PUR SONO NECESSARIE AL FUNZIONAMENTO DELLE WEB APPLICATION. COSA FARE QUANDO È NECESSARIO INSERIRE DEL CODICE PERSONALIZZATO?

Tutti sanno come funziona una tecnologia lato server. Un browser invia una richiesta ad un server web, quest'ultimo si occupa di interpretare le richieste contenute all'interno della pagina e scritte con un qualunque linguaggio lato server, ad esempio ASP.NET e produce una pagina HTML che viene rimandata al client. Il browser da solo non è in grado di interpretare una pagina scritta in PHP o ASP.NET, ma solo e soltanto puro HTML, con qualche eccezione. Al contrario non esiste un modo per il server di interagire con il browser. Per esempio non c'è modo per il server di accorgersi che l'utente ha digitato qualcosa in una casella di testo, oppure se ha ridimensionato la finestra del browser stesso. Certo, è sempre possibile effettuare dei postback, inviando la pagina al server come quando si compila il modulo e si fa clic sul pulsante Invia, ma ciò richiede continui reload e questo non è certamente possibile e accettabile in molti casi. Basti pensare all'evento "MouseMove", se ad ogni spostamento del mouse, si volesse informare il server della nuova posizione del puntatore, la pagina sarebbe certamente bloccata in continui postback, e non più utilizzabile. Per questo tipo di interazione, esistono per fortuna i cosiddetti script lato client, che è possibile scrivere in linguaggi appositi come JavaScript. Come dice il nome, gli script client-side sono eseguiti nel browser dell'utente e dunque possono reagire immediatamente alle azioni del mouse o della tastiera. Per esempio, utilizzando JavaScript, è possibile tener traccia della posizione del puntatore del mouse, e creare un effetto RollOver quando esso passa su un pulsante. In questo articolo si vedrà come utilizzare JavaScript, ed in generale i linguaggi di scripting lato client, nelle pagine ASP.NET, cioè in congiunzione ad una tecnologia lato server.

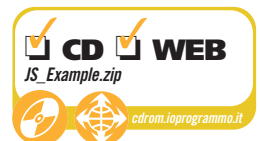
PRIMI PASSI CON JAVASCRIPT

Per mostrare come scrivere ed utilizzare un semplice script in una pagina ASP.NET ricorriamo ad un semplice esempio, ma che espone un'altra delle problematiche che non è possibile risolvere senza uno script lato client. Se volessimo realizzare una semplice casella di testo, che mostri l'orario sulla pagina, incontreremmo già alcune difficoltà. Potremmo agire lato server usando la classe *DateTime*, leggendo la proprietà *Now* per determinare l'orario attuale, l'ora ricavata e poi visualizzata sul browser ma in tal caso l'ora sarebbe quella del server, che potrebbe anche non coincidere con quella del computer client, ed anche se così, fosse si dovrebbe tener conto del lasso di tempo che intercorre tra la richiesta al server e la visualizzazione sul client, che potrebbe influire anche di diversi secondi sulla precisione dell'orario.

JAVASCRIPT VS ASP.NET

Quello che faremo in questo paragrafo sarà mostrare due soluzioni a confronto. La prima utilizzerà ASP.NET per prelevare la data lato server e mostrarla nel browser, la seconda utilizzerà JavaScript per prelevare la data dal client e mostrarla allo stesso modo nel browser. Ambedue sfrutteranno l'evento OnLoad ma in modo diverso. La pagina conterrà due TextBox, una per visualizzare l'ora del client, e la seconda per l'ora del server.

```
<asp:Label ID="Label1" runat="server" Text="Sono
le: "></asp:Label>
<asp:TextBox ID="txtClientTime" runat="server"
Width="250px"></asp:TextBox><br />
<asp:Label ID="Label2" runat="server" Text="ma sul
server sono le: "></asp:Label>
<asp:TextBox   ID="txtServerTime" runat="server"
Width="250px">
</asp:TextBox>
```

Per visualizzare l'ora attuale del server è sufficiente scrivere una riga di codice nel metodo Page_Load:

```
protected void Page_Load(object sender,
                                        EventArgs e)
{
   txtServerTime.Text=DateTime.Now.ToString();
}
```

Dal lato del client, invece, basta creare una semplice funzione javascript, inserirla ad esempio nel body della pagina, ed invocarla all'evento onload, inserendo il nome all'interno del tag body, come mostrato di seguito:

```
<body onload="ShowTime();">
...
<script type="text/javascript">
    <!--
        function ShowTime()
        {
      document.forms[0]['txtClientTime'].value=Date();
        }
    // -->
</script>
...
```

Se si prova ad eseguire la pagina, magari da un pc diverso dal server e con orari differenti, si vedrà come i valori delle due caselle di testo saranno, come ci si aspetta, diversi (vedi Fig.1).
Niente di nuovo, dunque, per chi ha già avuto modo di utilizzare JavaScript nelle pagine HTML classiche.

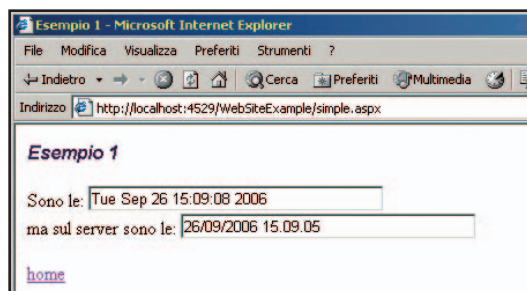



Fig. 1: **Orario sul client e sul server**

METODI E CLASSI

Ora, il punto è che se avessimo voluto passare un parametro dal codice lato client a quello lato server o viceversa avremmo avuto qualche problema. ASP.NET ed il framework .NET contengono però classi e relativi metodi che permettono di inserire codice JavaScript in una pagina in modo totalmente diverso.
L'idea è quella di far generare ad ASP.NET anche il codice JavaScript che deve essere utilizzato dal client, in tal modo le possibilità di interazione sarebbero sicuramente maggiori.

LA CLASSE CLIENTSCRIPTMANAGER

La classe ClientScriptManager è utilizzata per "creare dinamicamente" gli script lato client utilizzando codice lato server. Per ottenere un oggetto *ClientScriptManager* in una pagina web ASP.NET è sufficiente utilizzare la proprietà *ClientScript* dell'oggetto Page.
Per aggiungere uno script ad una pagina web, si possono utilizzare diversi metodi, la cui scelta dipende dal "quando" e "come" aggiungere gli script. La classe *ClientScriptManager* identifica univocamente gli script utilizzando una chiave di tipo String ed un Type. Gli script con la stessa chiave e lo stesso tipo sono considerati dei duplicati. Il tipo viene utilizzato per distinguere degli script nel caso in cui differenti controlli nella stessa pagina definiscano degli script con una stessa chiave. Esiste naturalmente il modo di verificare se uno script con una data chiave e tipo sia già stato registrato.
Il primo metodo utilizzabile per inserire uno script lato client in una pagina web è *RegisterClientScriptBlock*. Supponiamo di avere una pagina con due caselle di testo e di voler scrivere del codice javascript per elaborarne il contenuto, per esempio, per copiare il contenuto della prima nella seconda ma trasformando in maiuscolo tutte le lettere. La struttura della pagina potrebbe essere la seguente:

```
<asp:TextBox ID="txtSource" runat="server"
                               Width="250px" >
                          </asp:TextBox><br />
<br />
<asp:Button ID="btCopyServerSide" runat="server"
                   OnClick="btCopyServerSide_Click"
 Text="Copy Server-side" />
<input type="button" id="btCopyClient"
    runat="server" value="Copy Client-side" /><br />
<br />
<asp:TextBox ID="txtTarget" runat="server"
                Width="250px"></asp:TextBox><br />
```

La pagina contiene un pulsante btCopy ServerSide per eseguire l'operazione lato server, mediante il metodo seguente:

```
protected void btCopyServerSide_Click(object sender,
                                          EventArgs e)
{
   txtTarget.Text = txtSource.Text.ToUpper();
}
```

Per quanto riguarda invece il pulsante btCopyClient, nell'evento Page_Load verrà creato il codice JavaScript da eseguire al clic su di esso, registrato poi nella pagina mediante il metodo *RegisterClientScriptBlock*:

```
protected void Page_Load(object sender,
                                          EventArgs  e)
{
    if (!IsPostBack)

    {
        string idTarget = txtTarget.ID;
        string idSource = txtSource.ID;
        string scriptCopy = @"function Copy()
        {
document.forms[0]['"+idTarget+"'].value=
document.forms[0]['"+idSource+@"'].value.toUpper
                                          Case();
         };";

ClientScript.RegisterClientScriptBlock(this.GetType(),
                   "Script_Copy", scriptCopy, true);
btCopyClient.Attributes["onClick"]="Copy()";
    }
}
```

In questo caso viene utilizzato un overload del metodo *RegisterClientScriptBlock* con un parametro booleano, che indica se inserire o meno nel blocco di codice generato anche i tag <script> di apertura e chiusura del blocco.
Eseguendo la pagina potete verificare che il pulsante a destra effettua la copia e trasforma in maiuscolo il contenuto della prima TextBox, senza effettuare un postback della pagina sul server, cioè utilizza il codice javascript generato dinamicamente.
Se fosse necessario eseguire uno script lato client direttamente all'apertura della pagina, è possibile utilizzare il metodo *Register StartupScript* per registrare lo script da eseguire.
Il seguente esempio invoca la funzione *Date()* in maniera da ottenere la data attuale e scriverla in una TextBox appena la pagina viene caricata, quindi senza necessità di premere un pulsante o di forzare un altro evento:

```
protected void Page_Load(object sender,
                                          EventArgs e)

{
    string scriptShowTime =
 @"document.forms[0]['txtClientTime'].value=Date();
                                                   ";
    ClientScript.RegisterStartupScript(this.GetType(),
         "Script_ShowTime", scriptShowTime, true);
}
```

Una terza possibilità è quella di eseguire il codice lato client al submit di un modulo HTML, cioè non appena si verifica l'evento onsubmit.
Per registrare un blocco di codice in maniera che venga eseguita in tale occasione si può sfruttare il metodo *RegisterOnSubmitStatement*.
Il modo di utilizzo è analogo a quanto finora visto, dunque possiamo scrivere del codice per confermare o meno il submit della pagina così:

```
protected void Page_Load(object sender,
                                          EventArgs e)
{
    string script="return confirm('Are you sure you
                          want to submit this form?');"+
ClientScript.RegisterOnSubmitStatement(this.GetTyp
                        e(), "onsubmitscript", script);
}

protected void Button1_Click(object sender,
                                          EventArgs e)
{
    Button1.Text = "Submit effettuato";
}
```

In questo caso al submit della pagina, verrà mostrata una finestra di dialogo per confermare il submit, e solo se l'utente fa clic su 'Yes' verrà eseguito il codice del gestore *Button1_Click*.

IMPORTAZIONE DA FILE

In molti casi è necessario inserire in una pagina web, dei blocchi di codice javascript molto lunghi, quindi sarebbe comodo avere tali script salvati in uno o più file esterni e referenziarli in qualche maniera per mezzo di un url. Il metodo *RegisterClientScriptInclude* funziona in maniera simile a *RegisterClientScriptBlock*, ma legge lo script da un file esterno con estensione .js. Il file da includere verrà inserito prima di ogni altro script eventualmente presente, e dunque potrebbe non avere modo di accedere a qualche elemento della pagina.
Per provare il funzionamento del metodo, basta creare un file con del codice javascript, per esempio salvate la seguente funzione in un file external.js:

```
function Saluti(str)
{
    alert('Ciao '+str);
}
```

A questo punto, per registrare il file ed utilizzare la funzione Saluti, al click su un pulsante Button1, nell'evento Page_Load scriviamo:

```
protected void Page_Load(object sender,
 EventArgs e)
{
ClientScript.RegisterClientScriptInclude(this.GetType(
                    ), "externaljs",  "Scripts/external.js");
```

```
    Button1.Attributes.Add("onclick",
                          "Saluti('IoProgrammo')
     ");
}
```

Nella pagina generata verrà inserito il seguente riferimento:

```
<script src="Scripts/external.js"
                  type="text/javascript"></script>
```

Dunque il file external.js dovrà essere salvato in una directory Scripts della vostra applicazione web. Al clic sul pulsante Button1, verrà invocata la funzione Saluti con parametro “IoProgrammo”, e quindi il risultato sarà una Message Box lato client.

SCRIPT DI CALLBACK

Il codice lato client non può in generale comunicare con quello lato server. Per esempio una funzione javascript non può passare dei valori al codice della pagina durante un postback. Delle possibili soluzioni sono quelle di utilizzare campi Hidden, cookie, oppure come vedremo ora, implementando dei callback.
La funzionalità dei callback lato client è stata introdotta con ASP.NET 2.0, e permette di lavorare su una pagina, leggerne i valori e scriverli, senza postback, cioè senza dover rigenerare l'intera pagina. Tutto ciò viene chiamato anche callback fuoribanda (out of band), in quanto in tale situazione una funzione client side, per esempio in javascript, invia una richiesta asincrona ad una pagina, al di fuori del ciclo normale di vita. La pagina ASP.NET a questo punto esegue una versione modificata del proprio ciclo di vita e processa la richiesta. Per ottenere un riferimento alla funzione client che, una volta invocata, inizia il callback, è ancora una volta necessario utilizzare la classe *ClientScriptManager*, ed in particolare il metodo *GetCallBackEvent*
Reference. Come esempio vedremo come aggiornare una parte della pagina, senza effettuare alcun postback, e dunque con un'esperienza utente più piacevole ed efficace, e ciò apre nuove porte agli sviluppatori web, Ajax Docet!
La pagina HTML contiene una DropDownList con i nomi delle regioni italiane, ed un pulsante alla cui pressione si vuole ricavare l'elenco delle province appartenenti alla regione selezionata. Il tutto è facilmente realizzabile in ASP.NET, magari con un database contenente i dati, e quindi con una semplice query, oppure utilizzando un web service, in maniera del tutto trasparente. In questo caso però vogliamo che l'elenco di province appaia senza dare l'impressione che la pagina venga ricaricata, e dunque utilizzeremo il meccanismo dei callback, per aggiornare solo il codice HTML contenuto in un elemento div della pagina.
Come base dati utilizzeremo invece un database Access con due tabelle: Regioni e Province. Per interagire con il database si utilizzerà un oggetto AccessDataSource, semplicemente impostando la sua proprietà SelectCommand. Il metodo *GetProvince*, ricava l'elenco delle province di una data regione e crea una stringa contenente le province separate con un carattere |, perché è una stringa che la funzione di callback si aspetta:

```
public string GetProvince(string regione)
{
    try
    {
AccessDataSource1.SelectCommand="SELECT
  Province.Nome FROM Regioni INNER JOIN Province
                        ON Regioni.IDRegione =
                        Province.IDRegione WHERE
                   Regioni.Nome='"+regione+"'";
        DataView dv=
(DataView)AccessDataSource1.Select(
        DataSourceSelectArguments.Empty);
        DataTable dt = dv.Table;
        string province="";
        foreach (DataRow dr in dt.Rows)
        {
            province += dr[0].ToString() + "|";
        }
        if (province.Length > 0)
            province = province.Substring(0,
                            province.Length - 1);
        return province;
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

La Web Form, deve inoltre implementare l'interfaccia ICallbackEventHandler, che definisce due metodi:

```
void ICallbackEventHandler.RaiseCallbackEvent(
                          string eventArgument)
{
    _callbackArg = eventArgument;
}

string ICallbackEventHandler.GetCallbackResult()
```

```
{
        try
        {
            return GetProvince(_callbackArg);
        }
        catch (Exception ex)
        {
            throw new ApplicationException("Si è
        verificato un errore durante l'elaborazione della
                                richiesta:" + ex.Message);
        }
}
```

Il primo processa l'evento di callback inviato dal client, ricevendo un argomento stringa, in questo caso il nome della regione. Il secondo si occupa invece di invocare il metodo di accesso al database, GetProvince, e i restituire al client il risultato.
Dal lato client, la pagina web dovrà contenere due funzioni JavaScript, una che invierà il callback al server ed una per ricevere i risultati. La prima viene creata al primo caricamento della pagina, per mezzo del metodo GetCallback EventReference per ottenere un riferimento alla funzione callback, e poi utilizzando il già visto RegisterClientScriptBlock:

```
 if (!IsPostBack)
{
            string src =
     Page.ClientScript.GetCallbackEventReference(this,
            "arg", "DisplayResultsCallback", "ctx",
            "DisplayErrorCallback", false);
            string mainSrc = @"function
       GetProvinceOnServer(arg, ctx){ " + src + "; }";
Page.ClientScript.RegisterClientScriptBlock(
                                 this.GetType(),
"GetProvinceOnServer",  mainSrc,  true);
}
```

In questo caso la funzione di callback che verrà utilizzata dal server, una volta ricavati i risultati, sarà DisplayResultsCallback, scritta come mostrato qui di seguito:

```
function DisplayResultsCallback(result, context )
{
        var strHTML="";

        var s=result;
        if(result=!"")
        {
            var temp = new Array();
            temp=s.split("|");
            if(temp.length>0)
            {
                strHTML="Province:<br><ul>";
                for(i=0;i<temp.length;i++)
                {
                    strHTML+="<li>"+temp[i]+"</li>";
                }
                strHTML+="</ul>";
            }
            else strHTML += "<br><br><b>data not
                                        found</b>";
        }
        else
        {
            strHTML += "<br><br><b>data not
                                        found</b>";
        }

        divContents.innerHTML = strHTML;
}
```

La funzione precedente, una volta letto il risultato dal server, genera un elenco puntato, per mezzo dei tag html <ul> ed <li>, ed a questo punto lo visualizza inserendolo nell'elemento divContents.
Il diagramma in figura 3 mostra come interagisce il client con il server utilizzando la funzionalità di callback che utilizzeremo.
Il processo viene avviato quando l'utente seleziona una regione dalla casella a discesa e preme il pulsante di ricerca. Viene a questo punto invocato il metodo GetProvinceOnServer, creato dinamicamente sul server, in maniera da conoscere la funzione client di callback.
Ora il client genera l'evento di callback, utilizzando il metodo RaiseCallbackEvent, il server ricerca le province sul database, e le restituisce al client per mezzo di GetCallbackResult. Infine il client ottiene i risultati, e può occuparsi di aggiornare solo la parte di pagina che mostrerà i risultati.

Antonio Pelleriti

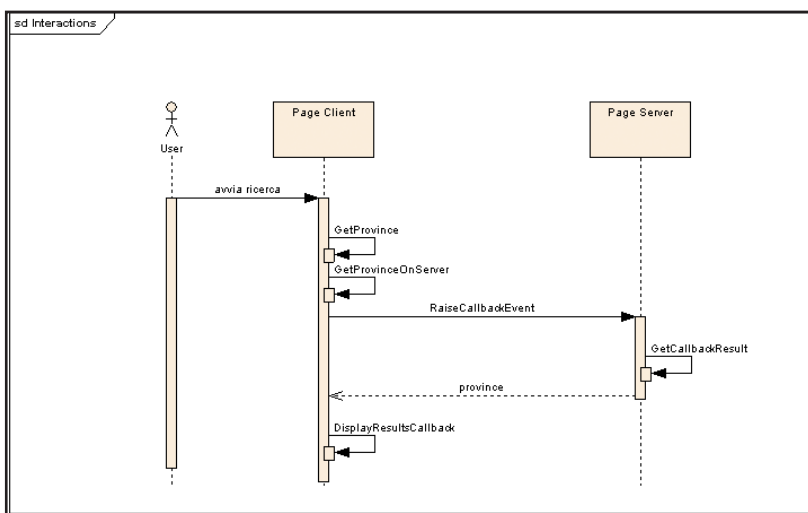



Fig. 3: ***Diagramma di sequenza durante i callback***

PAGAMENTI ONLINE: FACILE CON PAYPAL

IN UN SITO DI E-COMMERCE L'IMPLEMENTAZIONE DI UN MECCANISMO DI PAGAMENTO È UNO DEI PASSI CRUCIALI. TRA LE TANTE SOLUZIONI OFFERTE DAGLI ISTITUTI DI CREDITO ESAMINIAMO PAYPAL. VEDREMO COME SVILUPPARE UNA SOLUZIONE CON .NET

Da una parte ci siamo noi, per usare il termine tecnico il merchant, ovvero la nostra applicazione di E-Commerce dall'altra c'è un'applicazione residente nell'istituto di credito che contabilizza il pagamento della merce venduta nel nostro sito; sono due scenari paralleli che entrano in relazione attraverso un sistema di Payment Gateway. Tutto quanto riguarda il passaggio e l'accredito di contante non può risiedere nel database della nostra applicazione ma deve, necessariamente, essere collocato altrove, ovvero in un'applicazione parallela sviluppato ad hoc nell'ambito di un istituto di credito; Il nome in codice di questo software è Payment Gateway e, per intenderci, funziona come esposto nella Fig.1.

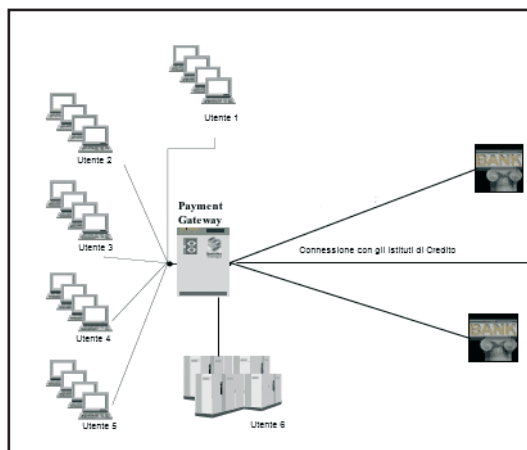



Fig. 1: **Schema del funzionamento di di un sistema di pagamento**

Banca Sella, PaschiInCommerce, TELEPay Light, OmniPay, Bank Pass, il nostro Paypal ed altri ancora offrono soluzioni di Payment Gateway che, per formalizzare il pagamento, rimandano ad una serie di Form collocati nel server dell'istituto di credito oppure, più raramente, inviano gli estremi del pagamento da un Form residente nella nostra applicazione. Ogni volta quindi che si tratta di implementare un meccanismo di pagamento elettronico bisogna interagire con la soluzione offerta da un istituto di credito che, in linea di massima, ripete lo schema appena esposto.

Scendendo nei dettagli, programmare una soluzione di Payment Gateway significa programmare i messaggi che intercorrono tra la nostra applicazione e quella residente nell'istituto di credito che, internamente, per quanto ci riguarda, rimane una sorta di scatola nera; In questo articolo sviscieramo il Payment Gateway offerto da Paypal nell'inedito porting verso .net 2.

PAYPAL VISTO DA VICINO

Abbiamo chiarito, in linea di massima, il funzionamento di un sistema di Payment Gateway;

La Fig. 2 illustra specificatamente quello implementato in Paypal

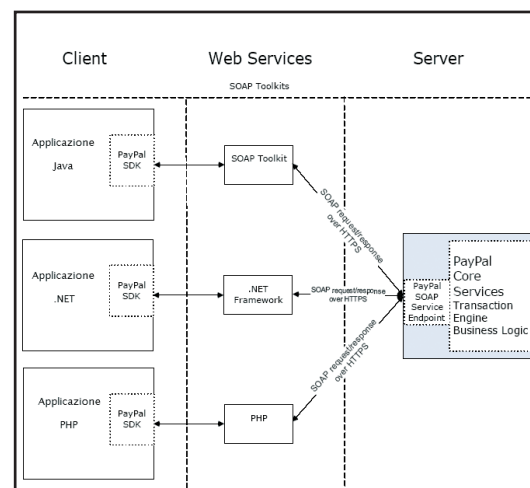



Fig. 2: **Architettura del sistema di payment di Paypal**

Analizzando la la figura scopriamo che i messaggi che intercorrono tra la nostra applicazione e Paypal viaggiano sul canale cifrato del protocollo HTTPS e sono descritti utilizzando Web Services ed i relativi messaggi SOAP;

Per familiarizzare con la programmazione delle API che sottendono a questi messaggi SOAP

Paypal mette a disposizione Kit di sviluppo dedicati alle tecnologie PHP, Java, Classic ASP, ed appunto .NET. Ufficialmente non è ancora implementata una soluzione .NET 2 e quella offerta per la "vecchia"' release è disponibile solo per C#; Nella 'scatola degli attrezzi' allegata all'articolo, sono disponibili i porting per VB.net e .NET 2, anzi sarà proprio su quest'ultima che ci rifaremo per gli esempi dell'articolo; nella prima pagina del SDK dobbiamo specificare se utilizzeremo il certificato allegato oppure se desideriamo usarne uno di nostra creazione. Dopo la selezione saremo inviati ad una pagina i cui collegamenti ipertestuali riassumono tutte le possibilità di programmazione delle API, ovvero tutti i possibili scenari di pagamento elettronico disponibili in Paypal; Scopriamole insieme

USIAMO L'API DODIRECTPAYMENT

Come allude il nome stesso questa API si occupa di gestire il pagamento della merce direttamente da una pagina collocata nel nostro sito;
La API mette in gioco il metodo *DoDirect PaymentRequestType* per inviare i dati relativi all'acquisto (carta di credito, IP dell'acquirente, tipologia della merce, etc) e gestisce la risposta attraverso *DoDirectPaymentResponseType* che ci fornisce dati sull'esito della transazione, ID Paypal della stessa, verifica della carta di credito etc Questa API è disponibile solo per il mercato statunitense, ed è anche quella di più facile implementazione. Dando una occhiata al codice della pagina *DoDirectPayment* non possiamo non notare

```
Imports com.paypal.soap.api
```

ovvero l'esplicito riferimento alla libreria Paypal
E, all'interno della Sub relativa al pulsante di pagamento

```
Protected Sub PayButton_Click(ByVal sender As
                              Object, ByVal e As EventArgs)

Dim api As PayPalAPI = New PayPalAPI()

Me.PayPalResponse = api.DoDirectPayment(parametri
                              ricevuti dai campi del Form)

End Sub
```

Gli altri mercati, compreso quello italiano, possono comunque utilizzare le API per Express Checkout e per l'interrogazione dello storico delle transazioni.

FACCIAMOLO CON EXPRESSCHECKOUT

La transazione effettuata con *ExpressCheckout* parte dalla nostra applicazione, viene perfezionata nel server di Paypal e termina tornando nel nostro sito; In breve:

1. il nostro cliente, già iscritto a Paypal, nell'ambito della nostra applicazione, dopo aver scelto i suoi acquisti, va in una pagina che riassume l'ammontare della spesa e preme il bottone "Express Checkout"
2. viene reindirizzato al sito Paypal nel quale, dopo essersi autenticato, rivede l'indirizzo al quale spedire la merce e seleziona il metodo di pagamento, al termine, viene diretto ancora al nostro sito attraverso la pressione del pulsante "Continue Checkout"
3. nella nostra applicazione controlla l'ordine e lo conferma utilizzando il bottone "Place order"

I tre passi che abbiamo descritto sopra prendono il nome di Integration Point. Nel primo Integration Point quando il cliente, preme il bottone 'Express CheckOut' dietro le quinte viene innescato il messaggio SOAP *SetExpressCheckout Request* verso Paypal, con i parametri *OrderTotal*, *ReturnURL*, *CancelURL* ed altri opzionali come il *MaxAmount*;
Paypal risponde con *SetExpressCheckoutResponse* che aggiunge sotto forma di coppia nome/valore il PayerId ed un token, ovvero una codice identificativo alfanumerico. A questo punto il browser del cliente viene reindirizzato al server Paypal passando sotto forma di stringa d'interrogazione il token ricevuto

```
https://www.paypal.com/cgi-bin/webscr?cmd=_express-
                    checkout&token=ValoreDelToken
```

nel server Paypal, dopo il login, è necessario specificare l'indirizzo al quale inviare la merce e le modalità di pagamento, nel medesimo tempo dal client parte il messaggio SOAP *GetExpressCheckout DetailsRequest* che richiede al server Paypal le informazioni relative all'indirizzo ed alla carta di credito gestite dal messaggio SOAP *GetExpressCheckout DetailsResponse* con i relativi parametri *PayerID*, *email address*, *shipping address* specificati in precedenza dall'utente. Se il cliente preme il bottone "Continue Checkout" viene rimandato alla nostra applicazione all'indirizzo specificato nel parametro ReturnURL che abbiamo visto precedentemente-
A questo punto incontriamo il secondo Integration Point. Questa pagina, nome in codice 'Order Review', elenca al cliente i dati del suo acquisto e, una volta premuto il bottone 'Confirm Order', invia il messaggio SOAP *DoExpressCheckoutPayment Request* con i parametri *Token*, *OrderTotal*,

PaymentAction e *PayerID* che provengono dal precedente *GetExpress CheckoutDetailsResponse*.
Il server Paypal risponde con *GetExpressCheckout DetailsResponse* che include il *TransactionID* ed altri elementi. Il cliente, finalmente, riceve una mail con i dati di conferma dell'acquisto; Quest'ultimo passo costituisce il terzo Integration Poin.
Per quanto riguarda il codice della relativa pagina ExpressCheckout nel SDK

```
Protected Sub CheckoutButton_Click(ByVal sender As
                        Object, ByVal e As EventArgs)

Me.PaymentAction = codice che controlla il tipo
              di pagamento a seconda del
              parametro passato dalla query string

End Sub
```

È necessario mettere a fuoco il metodo *PaymentAction* che innescherà il processo di pagamento *ExpressCheckout* appena descritto, ma con significative differenze se il parametro passato dalla stringa di interrogazione sia Sale, Authorization oppure Order; Sale indica che il pagamento viene concluso immediatamente, mentre viene differito se il parametro è Authorization oppure Order;

AUTHORIZATION & CAPTURE

Paypal prevede anche uno scenario di pagamento maggiormente flessibile, per intenderci immaginiamo di realizzare un sito di E-Commerce con merci da esportare in un altro paese.
Esistono fattori variabili, come il cambio della valuta, le spese di trasporto, l'eventuale indisponibilità del prodotto venduto ed altri ancora che "svaluterebbero", dopo un breve periodo di tempo, il prezzo fissato per l'acquisto.
Per ovviare a tutto questo Paypal ha previsto una tipologia di pagamento in cui il cliente ha la possibilità di ordinare la merce ed il venditore, in un secondo tempo, ha la facoltà di formalizzare l'acquisto ordinando il pagamento;
'Authorization Period' e' l'arco di tempo, fissato in 29 giorni, nel quale il venditore ha la facoltà di riscuotere il prezzo pattuito per l'acquisto (o parte dello stesso) dal conto PayPal o dalla carta di credito del cliente. Per "Honor Period" si intende il rinnovo dell'Authorization Period' nel quale PayPal garantisce al venditore che la somma stabilita per una transazione potrà ancora essere "catturato";
L'"Honor Period' e' fissato in 3 giorni, scaduti i quali PayPal non garantisce, pur facendo tutto il possibile, che recupererà il danaro dal cliente.
Quest< eventualità è stata prevista per cautelarsi verso acquirenti malintenzionati che, dopo aver effettuato un acquisto, bloccano il trasferimento di fondi dalla loro carta di credito o altro. Esiste anche la possibilità di annullare o rimborsare il pagamento. Concretamente, tutto questo viene implementato nelle pagine omonime del SDK

DoAuthorization
DoCapture
DoVoid

Nel dettaglio, se guardiamo il Code Behind della pagina DoAuthorization, ovvero della pagina nella quale il cliente opziona una merce

```
Imports com.paypal.soap.api

Public Sub SubmitButton_Click(ByVal sender As
                        Object, ByVal e As EventArgs)

Dim api As PayPalAPI = New PayPalAPI()
Me.PayPalResponse = api.DoAuthorization(parametri
                    ricevuti dai campi delWebForm)


End Sub
```

ritroviamo la linea di codice che importa la libreria delle API Paypal, la creazione di una nuova API Paypal ed il metodo *DoAuthorization()* che innesca il tipo di pagamento flessibile appena esposto;
Se diamo un'occhiata al relativo messaggio SOAP trasmesso scopriamo l'elemento
L'esito di questa transazione è gestito, in maniera non molto intuitiva, nella pagina *Authorization Response* dal metodo *DoAuthorizationResponse Type*

```
Protected ReadOnly Property Result() As
                        DoAuthorizationResponseType
        Get
        Return CType(Me.PayPalResponse,
                DoAuthorizationResponseType)
        End Get
End Property

Public Overrides Sub DataBind()
        If Me.IsTransactionSuccessful
                                        Then
                MyBase.DataBind ()
        End If
End Sub
```

Il codice essenziale della pagina nella quale il merchant gestisce la riscossione del prezzo è

```
Imports com.paypal.soap.api

```

```
Public Sub SubmitButton_Click(ByVal sender As
                            Object, ByVal e As EventArgs)

                    Dim api As PayPalAPI = New
                                            PayPalAPI()
                    Me.PayPalResponse =
                                        api.DoCapture()

End Sub
```

ed, ormai, dovrebbe essere facilmente comprensibile; per quanto riguarda l'annullamento dell'ordine scopriamo il metodo DoVoid

```
Public Sub SubmitButton_Click(ByVal sender As
                            Object, ByVal e As EventArgs)
                Dim api As PayPalAPI = New  PayPalAPI()
                    Me.PayPalResponse =
                  api.DoVoid(parametri ricevuti dai campi del
                                                WebForm)

End Sub
```

il cui esito viene gestito nella pagina *DoVoidResponse* attraverso le proprietà di AckCodeType

```
Private Sub Page_Load(ByVal sender As Object,
                                ByVal e As EventArgs

Me.VoidSuccessfulLiteral.Visible =
            Me.PayPalResponse.Ack = AckCodeType.Success
                    OrElse Me.PayPalResponse.Ack =
                        AckCodeType.SuccessWithWarning

End Sub
```

Nota che anche la pagina dedicata al DoDirectPayment Authorization si riferisce al tipo di pagamento differito Authorization & Capture

CONSULTARE LO STORICO DELLE TRANSAZIONI

Per finire è possibile effettuare una ricerca tra le transazioni effettuate con Paypal partendo dalla pagina TransactionSearch

```
Protected Sub SearchButton_Click(ByVal sender As
                            Object, ByVal e As EventArgs)

            Dim api As PayPalAPI = New PayPalAPI()
Dim startDate As DateTime = (parametri ricevuti dai
                                    campi del WebForm)
Dim endDate As DateTime =(parametri ricevuti dai
                                    campi del WebForm)

Me.Session(Constants.TRANSACTION_SEARCH_
                            SESSION_KEY) = api.
                    TransactionSearch(startDate, endDate)

End Sub
```

che, con il metodo *TransactionSearch*(), effettua la ricerca nello spazio circoscritto di due date; viene innescato poi un redirect nella pagina TransactionSearchResults:
se leggiamo il Code Behind di questa pagina scopriamo che viene creata una proprietà: PaypalResponse dalla API TransactionSearch ResponseType referenziata dalla session ripresa dalla pagina TransactionSearch vista prima

```
Protected ReadOnly Property PayPalResponse() As
                    TransactionSearchResponseType Get
Return
    CType(Me.Session(Constants.TRANSACTION_SEARCH
        _SESSION_KEY), TransactionSearchResponseType)
                    End Get

End Property
```

nel codice della Webform scopriamo un repeater che viene popolato dalle varie proprietà di PayPal Response: *PaymentTransactions*, TransactionID, *Timestamp*, *Status*, etc.

UN CASO CONCRETO

Chiarendo il funzionamento del SDK abbiamo compreso le funzionalità di base che caratterizzano di Paypal. Ora è venuto il momento di mettere a fuoco l'attenzione su casi concreti.
Nel paragrafo precedente abbiamo visto degli esempi a 'basso livello', dedicati alla programmazione delle API, ma le soluzioni migliori che Paypal offre sono semplici pezzi di codice HTML, creati al volo raccogliendo i dati che immettiamo in Form dedicati, pronti per essere 'copiati & incollati' nella nostra applicazione; per intendersi i tipici pulsanti:'Paga adesso', 'Carrello PayPal', 'Donazione'ed altri sono realizzati proprio in questo modo: tra queste soluzioni ve ne una che è discretamente diffusa ma che non viene mai trattata adeguatamente…
Immaginate un sito web nel quale è possibile prendere visione, dietro pagamento, di alcune sezioni private del sito, ma solo per un determinato periodo di tempo; si tratta di uno scenario tipico che Paypal risolve con una tecnica denominata di "Subscriptions and Recurring Payments". Per implementarla è appunto sufficiente copiare ed incollare il codice HTML generato dal sito Paypal;

Mentre vi rimando alla documentazione ufficiale per una informazione più dettagliata è comunque utile dare un'occhiata a questo codice

```
<form action="https://www.paypal.com/cgi-
        bin/webscr" method="post">
<input type="hidden" name="cmd" value="_xclick-
        subscriptions">
<input type="hidden" name="business" value="mail
        dell'account Paypal del Merchant">
<input type="hidden" name="item_name"
        value="Nome della sottoscrizione">
<input type="hidden" name="item_number"
        value="Identificativo numerico
        della sottoscrizione">
<input type="hidden" name="image_url"
   value="URL del proprio logo da conservarsi in un
   server https per evitare il messaggio "La pagina
   contiene oggetti protetti e non protetti" ">
<input type="hidden" name="no_shipping"
        value="1">
<input type="hidden" name="return"
value="URL della pagina del proprio sito alla quale
        andare dopo la conclusione positiva
        dell'abbonamento">
<input type="hidden" name="cancel_return"
value="URL della pagina del proprio sito alla quale
   andare qualora venga interrotta la procedura di
        abbonamento da parte dell'utente">
<!-- Parametri omessi relativi alla durata
        dell'abbonamento, alla rateizzazione,
        al rinnovo, etc.
        -->
<input type="hidden" name="custom" value="è
   possible impostare parametri personalizzati">

</form>
```

Naturalmente, durante la procedura di generazione del codice HTML, è possibile cifrare tutti questi parametri utilizzando il formato PKCS7 che è, tipicamente, quello utilizzato per la crittografia della posta elettronica.
Per gestire la risposta che Paypal invia verso la nostra applicazione al termine della procedura di "Subscriptions and Recurring Payments", ovvero l'esito positivo dell'abbonamento, la scadenza, la revoca e quant'altro, è possibile utilizzare Istant Payment Notification (IPN); per intendersi IPN è una risposta che il server Paypal invia ad una pagina dedicata nella nostra applicazione allo scopo di innescare le opportune operazioni di Back End, ovvero nel caso preso in esame, l'iscrizione dell'utente che ha pagato nel nostro database.
Scendendo nel dettaglio per implementare IPN è sufficiente accedere al proprio account Business o Premier nel sito Paypal e, nel proprio profilo, selezionare 'Modifica' dalla sezione 'Preferenze per Notifica immediata del pagamento' specificando poi l'indirizzo della pagina nella nostra applicazione al quale Paypal invierà il messaggio IPN (nota che il nome in codice di questo url è 'URL di notifica').
In alternativa, per i più smanettoni, è possibile aggiungere manualmente il campo notify_url nel codice HTML del nostro bottone specificando l' url dello script che gestisce IPN.
Una volta ricevuto il messaggio IPN la nostra applicazione, per evitare tentativi di truffa, deve inviare una risposta al server Paypal;
Il nome in codice di questa risposta è 'convalida della notifica' e può venire realizzata

- attraverso l'invio di una 'password condivisa' utilizzando una query string in un server SSL (ad esempio *https://www.miosito.com/UrlDiNotifica.aspx?password=pippo*)
- inviando un post

per motivi di brevità, ed anche di comodità, considerando la relatività difficoltà ed i costi nell'utilizzare un server SSL, analizziamo solo l'opzione che richiede l'invio del post.
In breve se volessimo creare un sito che offre contenuti a pagamento per i propri utenti utilizzando Paypal dovremmo

1. installare nella nostra applicazione il codice del bottone che innesca la procedura di "Subscriptions", copiato & incollato da Paypal
2. Creare una pagina che registri i nostri utenti nel database una volta che l'IPN ricevuto da Paypal e la successiva 'convalida della notifica' abbiano avuto un esito positivo;Nel dettaglio una pagina del genere potrebbe essere

```
Imports System.Net
Imports System.IO
Imports System.Text
Imports System.Web.Mail
Imports System.Collections.Specialized
Public Class SB_IPN
  Inherits System.Web.UI.Page
    Private Sub Page_Load(ByVal sender As
        System.Object, ByVal e As System.EventArgs)
                                Handles MyBase.Load
        'Put user code to initialize the page here
        Dim strFormValues As String =
                                Request.Form.ToString()
```

```
Dim strNewValue


' Crea la NOTIFICA DELLA CONVALIDA
Dim req As HttpWebRequest =
    CType(WebRequest.Create("https://www.
     sandbox.paypal.com/cgi-bin/webscr"), _
HttpWebRequest)

' imposta i valori per la
' NOTIFICA DELLA CONVALIDA
req.Method = "POST"
req.ContentType = "application/x-www-form-
                                urlencoded"
strNewValue = strFormValues +
                        "&cmd=_notify-validate"
req.ContentLength = strNewValue.Length

Dim stOut As StreamWriter = New
    StreamWriter(req.GetRequestStream(), _
Encoding.ASCII)
stOut.Write(strNewValue)
stOut.Close()

    'Inviata la notifica di convalida,
    'leggi il risultato

Dim strResponse As HttpWebResponse =
CType(req.GetResponse(), HttpWebResponse)
Dim IPNResponseStream As Stream =
                strResponse.GetResponseStream
Dim encode As Encoding =
  System.Text.Encoding.GetEncoding("utf-8")
Dim readStream As New
 StreamReader(IPNResponseStream, encode)


Dim read(256) As [Char]
' Reads 256 characters at a time.

Dim count As Integer =
               readStream.Read(read, 0, 256)

While count > 0
    ' Dumps the 256 characters
    ' to a string and displays the
    ' string to the console.
    Dim IPNResponse As New [String](read, 0,
                                    count)
    count = readStream.Read(read, 0, 256)

     ' Se la risposta alla 3
     ' di convalida è  VERIFIED
     ' tutto OK
     ' effettua l'aggiornamento del Database

    If IPNResponse = "VERIFIED" Then

AggiornaDatabase()

        Else' INVALID qualcosa
        ' non ha funzionato...
        ' Gestisci il problema

TroubleShooting()

    End Sub

    Sub AggiornamentoDatabase()

`codice per l'inserimento del database

    End Sub

    Sub TroubleShooting()

`codice per la gestione dell'errore

    End Sub
End Class
```

nota che questo codice, che opportunamente adattato, è stato generato da un sito web paypaltech.com che offre soluzioni per la maggior parte delle funzionalità di Paypal.
Per concludere questo paragrafo dedicato all'analisi dei casi concreti vi ricordo che per .NET 2 esiste uno Starter Kit con funzionalità di sito di E-Commerce al quale vi rimando senz'altro se volete vedere in azione un carrello pronto da 'copiare & incollare' nelle vostre applicazioni;
Non solo nel codice allegato all'articolo troverete uno Starter Kit rilasciato proprio da Paypal nel quale rivedere ancora le funzionalità descritte.

CONCLUSIONI

Le basi di Paypal che abbiamo analizzato in questo lungo articolo sono disperse nella monumentale documentazione fornita nel sito Paypal in tanti file .pdf che richiedono anche un non banale sforzo di traduzione; il mio lavoro deve essere inteso come un colpo d'occhio per iniziare a lavorare con questa utilissima piattaforma. Tuttavia le API esposte da paypal restano eccezionalmente semplici. Il sistema di ssicurezza esposto da paypal inoltre è uno dei più affidabili attualmente sul mercato. Se non volete spendere denaro in sistemi di POS elettronico Paypal è un'ottima soluzione

Luigi Corrias

JAVA E DATABASE PORTAFOGLIO FATTO

NEL PRECEDENTE NUMERO DI IOPROGRAMMO ABBIAMO PRESENTATO UNA TECNICA PER ESTRAPOLARE INFORMAZIONI FINANZIARIE DAL WEB. IN QUESTO NUOVO ARTICOLO MOSTRIAMO I METODI PER SALVARE I DATI IN SERIE STORICHE

Nello scorso numero di ioProgrammo avevamo gettato le basi per un'applicazione che una volta prelevate informazioni sull'andamento dei mercati via Internet, fosse in grado di elaborarle con fini statistici. Nel numero precedente avevamo concentrato la nostra attenzione sul reperiremento delle informazioni. In particolare avevamo utilizzato la pagina di yahoo finanza opportunamente parsarizzata. Avevamo utilizzatoassieme le API di Java per la rete e quelle per le espressioni regolari (RegEx). Infine ci eravamo anche posti il problema di come ottimizzare le interrogazioni su internet riducendo i tempi di attesa dovuti al protocollo TCP/IP. In particolare avevamo analizzato il fenomeno del TCP – slow start ed abbiamo così implementato un pool di thread ognuno del quale interroga una singola pagina (che rappresenta una singola quotazione) in maniera parallela a tutti gli altri.

...MA UN BACKEND È SEMPRE NECESSARIO

Appare ovviop che disporre di informazioni continuamente aggiornate risulta utile solo nel caso in cui abbiamo disponibile delle serie storiche con cui poterle confrontare. Un'altra esigenza abbastanza plausibile potrebbe essere anche quella di avere un proprio portafoglio in cui sono memorizzati dati come prezzo di carico dell'azione, numero delle azioni possedute etc...
Da queste brevi considerazioni si può subito intuire come, anche in questo caso, sia necessario avvalersi di un RDBMS. Visto quindi che un backend risulta indispensabile tanto vale andare a progettare un database che tenga conto anche degli aspetti "dinamici" della nostra applicazione Java. Ciò che si intende con quest'ultima affermazione è, ad esempio, la possibilità di memorizzare nel DB anche le stringhe di connessione per gli aggiornamenti su internet e delle tabelle che vengano continuamente aggiornate per produrre così delle serie storiche.

In figura 1 viene riportato il diagramma UML completo del database su cui ci baseremo. In realtà, vista la complessità dell'architettura, alcune tabelle rimarranno escluse da questo articolo, il che ci permette di concentrarci su un sottoinsieme del database completo. L'RDBMS di riferimento è il solito MySQL e potete trovare lo script per la creazione di tale db direttamente sul CVS di sourceforge (vedi box2). Naturalmente nulla vieta di usare un backend differente a patto che siate in possesso anche del relativo JDBC.

TUTTO INTORNO AD UN'AZIONE!

Come avrete avuto modo di notare dalla figura 1 al centro del diagramma compare la tabella azione. Questo perché essa rappresenta anche concettualmente la tabella centrale alla quale puntano la maggioranza delle chiavi esterne delle altre tabelle. Tale tabella è molto semplice ed è costituita da soli tre campi: *id, nome, descrizione*.
Come spesso si usa in questi casi l'id viene generato in maniera automatica, per cui bisogna solo inserire il nome dell'azione, ad esempio ENEL, ed opzionalmente una sua descrizione. Tutte le informazioni aggiuntive riguardanti tale azione verranno allocate in altre tabelle che avranno come chiave esterna proprio l'id della tabella azione. In particolare questo campo che funge da chiave esterna verrà chiamato azione_id.
Prendiamo ad esempio la tabella STORICO (vedi figura 1) e notiamo che essa, oltre che ad avere un proprio indice (id), contiene altri tre campi:

1) azione_id: il campo che contiene la reference ad AZIONE.id
2) prezzo: indica la quotazione dell'azione referenziata da azione_id
3) data: indica la data a cui si riferisce prezzo.

Si è inoltre imposto il vincolo che la coppia *azione_id* e *data* sia unica in modo tale che per ogni data non ci sia più di una quotazione.

Alla tabella AZIONE è anche collegata la tabella YAHOO dove è memorizzato il nome della connessione necessaria al robot per collegarsi al portale di yahoo finanza e recuperare le informazioni necessarie (vedi articolo precedente). In questo caso si è scelto di memorizzare tale link in una tabella differente proprio per separare concettualmente l'entità AZIONE dal particolare repository on-line di yahoo finanza. Potremmo infatti benissimo decidere in futuro di utilizzare un portale diverso da quello di yahoo, ma l'azione a cui ci riferiamo rimarrebbe pur sempre la stessa. In quel caso quindi non dovremmo far altro che aggiungere un'altra tabella ALTRO_PORTALE che abbia una chiave esterna che punti alla solita "tabella centrale" AZIONE. Ad una situazione del tutto analoga è riconducibile la relazione che intercorre fra le tabelle INDICI e STORICO_INDICI. Questa volta però tali tabelle conterranno informazioni relative agli indici di borsa, tipo MIBTEL, MIB30 etc...
Come avrete notato dal diagramma, l'unica differenza rispetto al caso precedente è che la stringa di connessione a yahoo finanza viene memorizzata sulla stessa tabella INDICI. Questo è stato fatto per ragioni puramente di ordine pratico. Ad ogni modo quest'ultima soluzione non preclude minimamente la possibilità di aggiungere eventualmente in futuro ulteriori tabelle che puntino ad altri portali come ipotizzato precedentemente. Per ragioni di spazio non ci soffermeremo su tutta la struttura riportata in figura 1, anche perché tra il diagramma UML e lo script SQL allegato alla rivista potrete sicuramente orientarvi benissimo. Ciò su cui invece è necessario focalizzare la nostra attenzione è collegarci efficacemente al database tramite un JDBC e come tenere ben separata la logica concernente l'interazione con il DB da quella riguardante più il lato applicativo.

GESTIAMO LA NOSTRA CONNESSIONE

In figura 2 è riportata l'architettura java per la connessione ad un RDBMS così come è riportata nella documentazione ufficiale della SUN. Prestando attenzione alla parte destra di tale illustrazione, cioè focalizzandoci sui JDBC e tralasciando la parte concernente gli ODBC, notiamo come esista un'entità chiamata *DriverManager* che si occupa di "tirare su" l'opportuno driver per l'RDBMS desiderato. In realtà tale classe (perché DriverManger alla fine non è altro che una classe) può fare varie altre cose oltre a quelle di richiamare uno specifico JDBC; tanto per dirne una può elencare tutti i driver registrati dall'applicazione cosicché si potrebbe ad esempio pensare di realizzare una GUI in cui l'utente può scegliere a quale backend collegarsi.
Per ora a noi interessa solamente capire quando e come aprire e chiudere una connessione verso il nostro database. A questo scopo è stata creata la classe MyDBConnection che ci permetterà di eseguire tali operazioni in maniera abbastanza agevole. Al suo interno incapsula un attributo Connection che, è bene ricordarlo, è solamente un'interfaccia prevista dal package java.sql.

```
public class MyDBConnection {
    private Connection myConnection;
            ....
```

Una volta istanziata questa classe, per stabilire una connessione con il database è necessario invocare il metodo init che è così strutturato:

```
            try {
                Class.forName("com.mysql.jdbc.Driver");
```

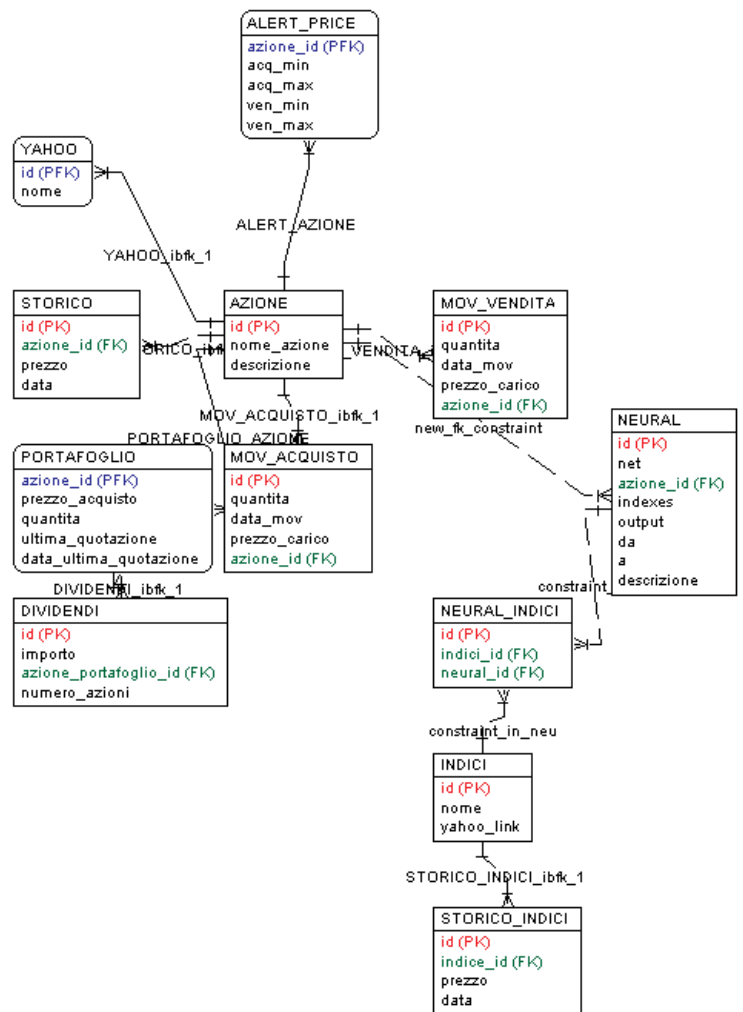



Fig. 1: ***Diagramma UML con database implementato.***

NOTA

MARKAD SU SOURCEFORCE

Il progetto si chiama Market Advisor e lo potete trovare all'indirizzo https://sourceforge.net /projects/markad. Approfitto della disponibilità di questa rivista e del suo direttore Fabio Fanesi per rivolgere un invito a tutti i lettori che vogliano unirsi a tale progetto di contattarmi via email. Abbiamo bisogno di varie figure, oltre a sviluppatori Java, servono anche appassionati di reti neurali, analisti economici, web designer per realizzare la home page e forse anche esperti di MySQL che magari permettano all'applicazione di puntare su un db comune. Grazie a tutti

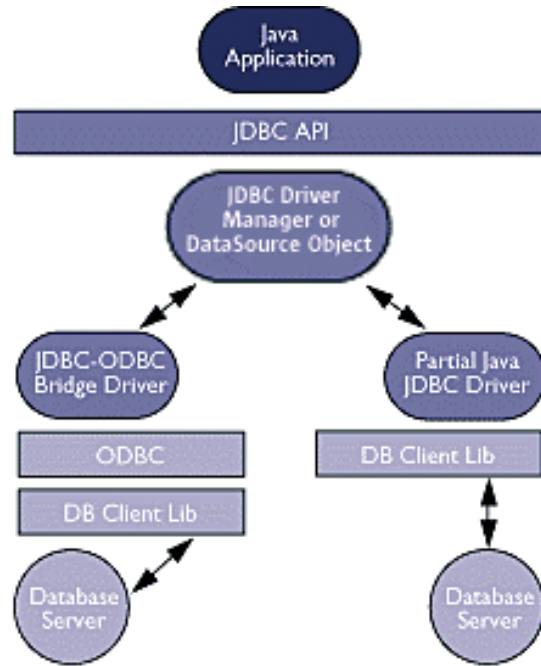



Fig. 2: ***Architettura java per una connessione ad un RDBMS***

```
} catch (ClassNotFoundException e)  {
/ System.err.println(e.getMessage());
}
        Properties prop = new Properties();
        prop.setProperty("user","***");
        prop.setProperty("password","***");

prop.setProperty("zeroDateTimeBehavior"
                              ,"convertToNull");
    myConnection=DriverManager.getConnection(
        "jdbc:mysql://localhost/stock_quote",prop);
    myConnection.setAutoCommit(false);
```

Per prima cosa bisogna cosa bisogna caricare dinamicamente la classe Driver mediante il metodo statico Class.forName, come si vede da:

```
try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        System.err.println(e.getMessage());
}
```

tale metodo va a cercare la classe specificata nel suo argomento nel classpath della vostra JVM, quindi bisogna ricordarsi naturalmente di impostare il classpath in modo tale che esso comprenda anche il jar del JDBC desiderato (nel nostro caso utilizziamo quello di MySQL). Così facendo il driver istanziato si registrerà nel DriverManager e sarà quindi pronto per fornire una corretta connessione al database.
Le sintassi possibile per ottenere un oggetto Connection sono molteplici. Quella da me considerata (per quel che conta) la più pulita, è quella che fa uso di un oggetto Properties, che non è altro che una classe derivata da HashMap; per questo essa funziona con il classico paradigma chiave-valore. Nel nostro caso, oltre che lo user e la password, abbiamo specificato:

```
prop.setProperty("zeroDateTimeBehavior",
                              "convertToNull");
```

Questa proprietà fa si, che per quei valori di tipo Date o Datetime, che nelle tabelle del database non sono specificati e che quindi vengono impostati tipicamente come 0000-00-00 di default, vengano poi tradotte in java in classi null. Se si omettesse questa impostazione ad una data del tipo sopra citato l'applicazione java lancerebbe un'eccezione. Per ottenere finalmente un oggetto Connection ora non resta da fare altro che eseguire il metodo

```
    myConnection=DriverManager
.getConnection(
"jdbc:mysql://localhost/stock_quote",prop);
```

L'ultima riga indica invece che abbiamo scelto un backend che supporti le transazioni, infatti nel nostro caso abbiamo scelto l'engine InnoDB per tutte le nostre tabelle. Tramite il metodo getConnection è poi possibile recuperare l'oggetto Connection opportunamente instanziato.
Per un suo utilizzo immediato è stato implementato un metodo statico executeQuery:

```
public static ResultSet executeQuery(String query)
throws SQLException {
            MyDBConnection
            myc = new MyDBConnection();
            myc.init();
            Statement s =
myc.myConnection.createStatement(ResultSet.TYPE_FOR
WARD_ONLY, ResultSet.CONCUR_READ_ONLY);
            ResultSet rs =
                    s.executeQuery(query);

            return rs;
    }
```

L'unica cosa che tengo a far notare in questo caso è che la connessione questa volta non viene chiusa al termine dell'esecuzione della query. Questo perché renderebbe impossibile al chiamante di leggere i dati contenuti in ResultSet; sarà quindi onere del chiamante stesso di invocare il metodo close di rs.
Nel prossimo paragrafo vedremo come questo tipo di pratica sia vivamente sconsigliata.

TRA STORED E SEMPLICI QUERY

Come avrete già capito quest'articolo, oltre che a presentare una panoramica sul backend di MarkAd, vuole anche fornire le nozioni base per progettare un'applicazione Java che interagisca con un RDBMS. Abbiamo già visto che il metodo excuteQuery presenta un primo punto debole rappresentato dal fatto che la chiusura della connessione è affidata alla buona volontà del chiamante. Naturalmente il rischio collegato a quest'aspetto è abbastanza alto, sia perché non stiamo fornendo una procedura "atomica", cioè una procedura che inizi e finisca all'interno del metodo stesso, sia perché, nel caso di più client, avremmo più connessioni "appese" senza poter sapere quando e se verranno chiuse (con conseguente decadimento delle prestazioni dell'applicazione e del RDBMS). Sotto l'aspetto più proprio dell'ingegneria del software c'è inoltre un fatto che ci dovrebbe fare rabbrividire ancor di più. Immaginiamo di utilizzare tale metodo ogni qual volta volessimo effettuare un'interrogazione verso il database. Oltre ai problemi sopra esposti prima andremmo incontro anche ad una proliferazione di codice SQL sotto forma di stringa all'interno del nostro codice Java. Basterebbe una piccola modifica o sui requisiti o sul backend stesso e non sapremmo più dove mettere le mani! Una delle possibili soluzioni in questo caso è l'utilizzo delle stored procedures. Esse sono dei "programmini" scritti e ospitati nel RDBMS stesso ed invocabili poi dall'applicazione. Questo tipo di soluzioni è molto adottata laddove la Business Logic è affidata a chi progetta il DB. Noi invece abbiamo deciso di utilizzare una strategia che è una via di mezzo tra le due fin qui presentate. Il primo obiettivo da prefiggersi è quello di centralizzare e riutilizzare il codice il più possibile. La API della SUN prevedono un'interfaccia che prende il nome di PreparedStatement che se ben usata si rivela uno strumento molto efficace in termini di riutilizzabilità ed incapsulamento del codice. Vediamo subito un semplice esempio concreto per capire come funzionano questi oggetti.Supponiamo di voler recuperare l'id di un'azione avendo a disposizione il nome della stessa (vedi figura 1). Di primo acchito verrebbe spontaneo scrivere uno statement SQL di questo tipo:

```
"SELECT id FROM AZIONE WHERE
AZIONE.nome_azione="+azione+";".
```

Questo sarebbe poi "dato in pasto" a quel metodo orrendo che abbiamo prima deprecato.
Oltre che ad essere un modo un po' macchinoso per creare delle query, vedremo più avanti che per query maggiormente complesse questa soluzione provocherebbe solo forti emicranie senza portare alcun vantaggio; sarebbe inoltre anche auspicabile creare degli statement più "robusti". Ad esempio effettuare un "check" sui tipi sarebbe una feature molto comoda, cioè poter stabilire a priori se il campo della query sia un INT, VARCHAR, BLOB etc...
Vediamo ora il codice necessario per implementare la query precedente ed incapsularla in un metodo facilmente riutilizzabile da qualsiasi punto del programma. Innanzitutto tale metodo dovrà avere un argomento di tipo String (il nome dell'azione) e restituire un int (l'id dell'azione).

```
public class GeneralProcedures {
        private MyDBConnection con =
                new MyDBConnection();
...
        private static final String getAzioneID =
                "SELECT id FROM AZIONE WHERE
                AZIONE.nome_azione=?";
....
        public int getAzioneID(String nome) {
                return
        getAzioneID(con.getMyConnection(), nome);
        }
        public static int getAzioneID(Connection
                        c, String nome) {
                try {
                PreparedStatement ps =
c.prepareStatement(getAzioneID);
                        ps.setString(1, nome);
                        ResultSet rs =
                        ps.executeQuery();
                        if (rs.next())
                          return  rs.getInt(1);
                        else
                                return -1;
                } catch (SQLException e) {
                        return -1;
                }
        }
```

Cosa abbiamo fatto? Prima di tutto è stata definita una stringa costante e statica in cui è stata definita la query ed il parametro da settare che viene identificato con un punto interrogativo. Il metodo getAzioneID non fa altro che generare un statement dalla stringa precedentemente definita attraverso una connessione:

```
PreparedStatement ps =
c.prepareStatement(getAzioneID);
```

poi settiamo il parametro nome ed eseguiamo la query:

```
                        ps.setString(1, nome);
                        ResultSet rs =
```

L'AUTORE

Laureato in ingegneria elettronica presso l'università Politecnica delle Marche, lavora presso il reparto R&D di un gruppo multinazionale giapponese leader nel campo delle tecnologie di sistemi real-time. Nei limiti della disponibilità di tempo risponde all'indirizzo andreagaleazzi@fsfe.org

```
ps.executeQuery();
```

Infine se l'azione è stata trova se ne restituisce l'id, altrimenti si restituisce il valore -1 per indicare che qualcosa è andato storto. La sintassi per definire i parametri, come avete visto, è molto semplice: basta invocare il metodo set opportunamente tipizzato (setString, setInt etc...) che ha come argomenti, oltre che il valore, l'indice del punto di domanda che si intende andare a sostituire (nel nostro caso ne avevamo solamente uno). Per chiarire meglio quanto sia conveniente seguire questo tipo d'approccio implementiamo una procedura che ci permetta di visualizzare come è distribuito il nostro portafoglio azionario in un grafico a torta. Definiamo una classe PortafoglioProcedures dove implementare tutti i metodi che operano sulla tabella PORTAFOGLIO (ricordate? Centralizzare il codice!). Ciò che dobbiamo fare in questo caso è moltiplicare il prezzo di carico per il numero di azioni detenute e memorizzare tale valore in un PieDataSet della libreria JfreeChart (vedi box 1).
Prima di tutto prendiamo tutte le azioni presenti nel portafoglio con una semplice query:

```
private static final String queryAzione="SELECT
id,nome_azione FROM AZIONE";
```

Eseguita tale query dobbiamo vedere quanto ogni azione "pesi" all'interno del portafoglio:

```
private static final String totQuantita="SELECT
PORTAFOGLIO.quantita FROM"+
"  PORTAFOGLIO WHERE PORTAFOGLIO.azione_id=?
"+ "GROUP BY PORTAFOGLIO.azione_id";
.......
public static PieDataset posDeposito() {
                    DefaultPieDataset ds = new
                                DefaultPieDataset();
                    MyDBConnection con = new
                                MyDBConnection();
                    try {
                                con.init();
                    ...
          while(res.next()){
                    int id=res.getInt(1);
                    PreparedStatement ps
                                con.getMyConnection().
                    prepareStatement(totQuantita);
                                ps.setInt(1,id);
                                          ResultSet
                    resTot =ps.executeQuery();

if(resTot.next()){
          double carico=prezzoCarico(id);
          int tot=resTot.getInt(1);
          ds.setValue(res.getString(2),carico*tot);
}
          else
 System.err.println(res.getString(2)+" non trovata");
}
```

Fig. 3: **Grafico a torta che mostra la ripartizione di un ipotetico portafoglio azionario**

Per brevità ho riportato solo la parte più significativa del codice. Per ogni azione creiamo un PreparedStatement in cui settiamo l'id dell'azione stessa; eseguiamo poi la query per ottenere il prezzo di carico ed il numero totale di azioni detenute:

```
double carico=prezzoCarico(id);
int tot=resTot.getInt(1);
ds.setValue(res.getString(2),carico*tot);
```

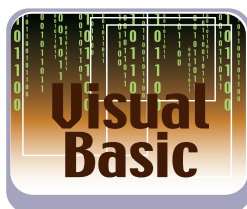
Moltiplicando questi due fattori si ottiene il carico dell'azione rispetto al portafoglio.
Purtroppo non c'è spazio in quest'articolo per approfondire anche la libreria JfreeChart, speriamo di poterlo fare in uno dei prossimi articoli. Ad ogni modo il risultato finale di questo metodo (con azioni e prezzi di carico assolutamente casuali) lo potete osservare in figura 3.

CONCLUSIONI

In quest'articolo siamo partiti con l'analizzare il back-end di markad e abbiamo finito con l'analizzare come si possa strutturare in maniera corretta del codice che si interfacci spesso con un RDBMS. Lo scopo era fornire le basi per l'ottima comprensione del collegamento di Java ai database. Nasturalmente tutte le tecniche vanno adattate alle proprie esigenze, ma in linea del tutto generale, quanto mostrato fin qui, rappresenta un ottimo "template"

Andrea Galeazzi

CREIAMO ADDIN PER VB ED OFFICE

SCOPRIAMO COME FUNZIONANO, COSA SONO E COME SI DIVIDONO GLI ADDIN. COME ESEMPIO PRATICO IMPLEMENTEREMO UN'ESTENSIONE DI VISUAL BASIC CHE CONTERÀ IL NUMERO DI CONTROLLI PRESENTI IN UNA FORM

Un *AddIn* è un componente che aggiunge nuove funzionalità ad un'applicazione, detta applicazione *Target*. Queste funzionalità, in genere, sono racchiuse in una *DLL*. In questo articolo presentiamo gli *AddIn* per l'*IDE* di *Visual Basic* e per le applicazioni di *Microsoft Office*. Nel caso di *Visual Basic*, possiamo affermare che, uno dei principali obiettivi degli *AddIn* è quello di automatizzare dei processi difficoltosi, ripetitivi, noiosi, che richiederebbero molto tempo e risorse se eseguiti manualmente, per esempio la creazione di *moduli di classi*, la gestione delle API di *Windows*, la gestione dei *file di Risorse*, ecc. Nell'*IDE* di *VB* per caricare/scaricare gli *AddIn* è presente l'*AddIn Manager* attivabile dalla voce di *Menu AddIns* (*Aggiunte*). In *Microsoft Office*, invece, oltre agli *AddIn* implementati con il supporto degli elementi *COM* (*Component Object Modul*), detti appunto *Addin COM*, è possibile usare gli *AddIn* implementati utilizzando i modelli forniti dalle varie applicazioni *Office*, per questo si parla di *AddIn Word*, *Excel*, ecc. Nel corso dell'articolo presenteremo i seguenti argomenti.

- Come creare un *AddIn*, utilizzando il modello di progetto fornito da *Visual Basic*.
- Che cos'è e come opera l'oggetto *AddInDesigner*, oggetto alla base dei *progetti AddIn*.
- Come creare un *AddIn* per *Office*.

Inoltre faremo una panoramica su alcuni elementi, necessari per lo sviluppo quali l'interfaccia *IDTExtensibility2*, i *file di Risorse, la gestione degli eventi ecc.* Come esempio introdurremo un *AddIn* che permette di contare e classificare i controlli presenti sulle *Form* di un progetto.

VB ED OFFICE QUALI ADDIN?

Il progetto *AddIn* selezionabile dalla maschera *nuovo progetto* di *Visual Basic* è costituito da un *Activex DLL* con una Form e un oggetto *AddInDesigner*. Quest'ultimo, permette di ottimizzare il processo di registrazione e di gestione delle varie funzionalità fornite dall'AddIn, che diversamente bisognerebbe gestire con interfacce di basso livello quali IDTExtensibility2, *IDTWizard ecc*.

I tipi di *AddIn* implementabili con *Visual Basic* sono: *AddIn Generico, Wizard, Utility* e *Builder*. Questa distinzione è fatta in base al comportamento e al tipo d'interfaccia supportata. L'*AddIn Generico*, è un *Activex DLL o EXE* che supporta l'interfaccia *IDT Extensibility2*.

Il *Wizard* è un *Activex* che supporta l'interfaccia *IDTWizard* e che ha lo scopo di guidare l'utente, passo-passo, attraverso un processo. Un *Wizard*, in genere, è composto da più *frame* (uno per ogni passo).

Un *Utility*, invece, è un *AddIn* compilato come *Activex* eseguibile e che quindi non necessariamente richiede un ambiente di sviluppo per essere avviato.

Un *Builder* è un tipo di *AddIn* che permette di controllare certi eventi dell'*IDE*, come inserimento di un nuovo componente, aggiunta di un nuovo controllo alla *Toolbar*, ecc.

In base a questa breve descrizione s'intuisce che alcuni tipi di *AddIn* non sono visibili, ma girano in *background*, e rispondono ad alcuni eventi. Per quanto riguarda *Office* le *Aggiunte COM* sono contenute, anch'esse, in file con estensione *DLL* o *EXE* che possono essere create con *Visual Basic*, con un altro prodotto di *Visual Studio*, con *Microsoft Office Developer* oppure con prodotti di terze parti. Gli *AddIn non COM*, per le applicazioni *Office*, invece, possono essere creati con i modelli di *AddIn* fornite dalle stesse applicazioni *Office*.

Per esempio con *Excel* è possibile implementale un *AddIn* che in seguito può essere caricato da tutte le altre *Cartelle Excel, ecc*.

Conoscenze richieste
Conoscenze di base sulla gestione dei progetti e dei controlli di Visual Basic.

Software
Piattaforma Windows 2000 o superiore - Visual Basic 6 SP6.

Impegno
Tempo di realizzazione

COSTRUIAMO UN ADDIN GENERICO

Il modello di progetto *AddIn Generico* permette di ottimizzare il processo di registrazione dell'*AddIn* e crea il supporto per gestire le varie funzioni che l'*AddIn* dovrà fornire. Per creare un nuovo progetto *AddIn* di tipo generico, dalla finestra *Nuovo Progetto* è necessario selezionare l'icona *AddIn*. In questo modo viene creato un progetto con una *Form* nominata *frmAddIn* e un oggetto *AddInDesigner* nominato *Connect*. La *form* contiene due CommandButton: *Ok* e *Cancel*.

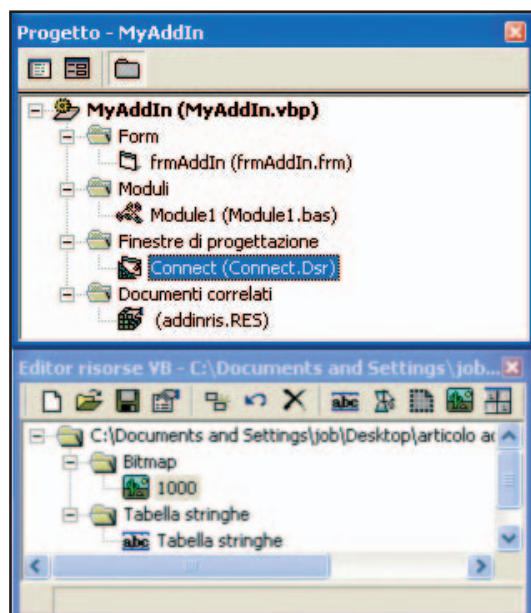



Fig. 1: **L'albero progetto del nostro esempio.**

L'*AddInDesigner* è il modulo che permette di impostare le principali proprietà dell'*AddIn*. Le proprietà

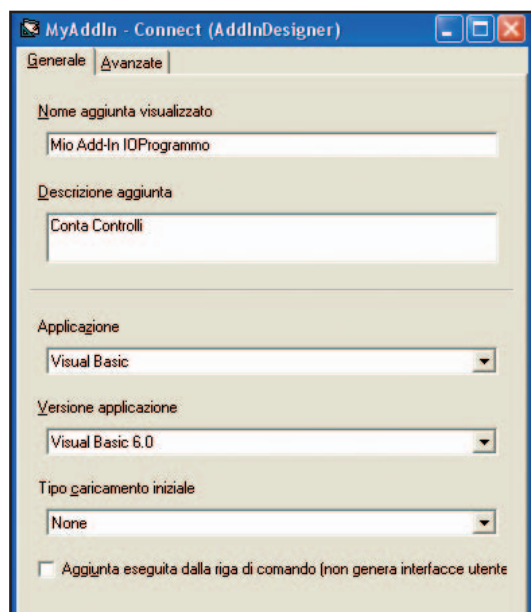



Fig. 2: **La maschera dell'AddInDesigner.**

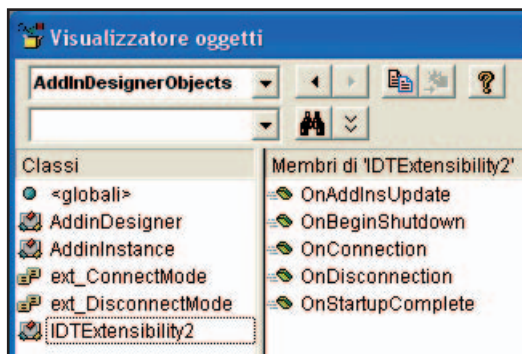
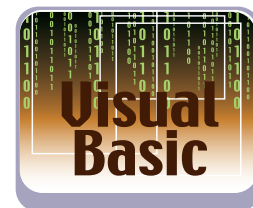
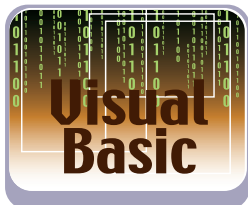



Fig. 3: **AddInDesignerObject nel Visualizzatore di Oggetti**

sono distribuite su due *schede*: *General* e *Advanced*. Sulla *General* si possono impostare le seguenti proprietà: *Nome, Descrizione, Applicazione Target* (*IDE, Word, Excel* ecc.), *versione applicazione, tipo di caricamento* (*None, On Startup, Command Line, Command Line/On Startup*) ecc. Nella scheda *Advanced* invece è possibile specificare: il *file DLL* contenente delle risorse per l'*AddIn*, una Key del registro che conterrà le stringhe specificate nella ListView sottostante (nominata Addin Specific Data), ecc.

L'INTERFACCIA IDTEXTENSIBILITY2

Gli *AddIn Generici*, come accennato, sono degli *ActiveX Server Component* che implementano l'interfaccia *IDTExtensibility2*.
La *IDTEXTENSIBILITY2*, in particolare è l'interfaccia che contiene i metodi per gestire le operazioni sugli AddIns. IDTExtensibility2 è contenuta nella libreria VB6EXT.OLB (Microsoft Visual Basic 6.0 Extensibility).
Quando si usa il progetto AddIn Generico, fornito da Visual Basic, essa è implementata automaticamente nel controllo AddInDesigner (controllate nei riferimenti del progetto AddIn). I metodi esposti da IDTExtensibility2 sono i seguenti:

1. OnAddinsUpdate, è il metodo invocato quando la collezione degli AddIn (nominata AddIns) è modificata.
2. BeginShutdown, è richiamato prima della chiusura dell'ambiente di sviluppo.
3. OnConnect, è richiamato quando l'ambiente di sviluppo carica l'AddIn, esso connette l'AddIn all'Extensibility Object Model.
4. OnDisconnection, è richiamato quando l'AddIn è scaricato.
5. OnStartupComplete, è richiamato quando è completato lo Startup dell'IDE.

GLI OGGETTI ADDIN E ADDINS

Le aggiunte nell'ambiente di sviluppo sono rappresentate dagli oggetti *AddIn* e *AddIns*, a cui si può fare riferimento direttamente dall'oggetto di primo livello *VBIDE*. AddIn fornisce informazioni sulle Aggiunte, la collezione AddIns invece contiene i riferimenti agli *AddIn* accessibili. Ogni *AddIn*, gestito da *AddIn Manager*, ha un riferimento in questa collezione. In particolare utilizzando la collezione si possono *connettere/disconnettere* gli *AddIn*, oppure, soltanto, elencarli, come mostreremo più avanti con gli esempi.
Un progetto *AddIn COM* è strutturalmente analogo all'*AddIn Generico*, infatti, anch'esso è basato sull'oggetto *AddInDesigner*. Le differenze, però, sono nell'uso di oggetti specifici della piattaforma *Target*. In particolare l'*AddIn COM* non utilizza dichiarazioni come *VBInstance As VBIDE.VBE*, compatibili con l'*IDE* di *Visual Basic*, ma dichiarazione come *AppInstance As Object*, compatibile con *Office*. Dove *AppIstance* sarà un oggetto *Application*, cioè un'applicazione *Word, Excel*, ecc. In particolare la scelta dell'applicazione *Office*, cui l'*AddIn* è dedicato, bisogna farla sull'*AddInDesigner* attraverso la proprietà Application.
Questo, però, non è un vincolo poiché non ha effetto sul codice all'interno dell'*AddInDesigner* che è implementato in modo generico. Per implementare un *AddIn COM* per *Word* devono essere seguite le seguenti linea guida:

1. il progetto AddIn deve referenziare le librerie di Office (Microsoft Office 9.0 Object Library o superiore);
2. l'AddInDesigner deve essere impostato in modo che all'atto della compilazione venga creato un AddIn Word;
3. nell'AddInDesigner deve essere predisposto il codice per interfacciarsi con l'architettura di programmazione di Office;

Inoltre bisogna prevedere altre dichiarazioni tra le quali segnaliamo:

```
Dim MenuCommandBar As
                    Office.CommandBarButton
Public WithEvents GestoreEventi  As
                    Office.CommandBarButton
```

Dove *MenuCommandBar* è di tipo *Office.CommandBarButton* e serve per creare un riferimento verso il *ComandBarButton* inserito nell'applicazione *Office Target*; *GestoreEventi*, invece, serve per catturare le azioni fatte sulla voce di menu associata a *MenuCommandBar*.

ADDIN OFFICE NON COM

Per costruire un *AddIn Office* non *COM* basta salvare un documento *Office* in un particolare formato. Per esempio per creare un *Excel AddIn* bisogna procedere nel seguente modo.

1. Creare un nuovo *Workbook* (*Cartella di lavoro Excel*).
2. Implementare, tramite *Visual Basic Editor*, il codice che l'*AddIn* dovrà gestire.
3. Definite il nome (*Title*) dell'*AddIn* che dovrà comparire nella finestra delle *Aggiunte Excel*, questo per esempio si può fare selezionando Proprietà (*Properties*) dal menu file di *Excel*.
4. Compilare il progetto attraverso il *Menu Debug di Visual Basic Editor*.
5. Salvate il *Workbook* nel formato *Aggiunta Excel* che è un file con estensione *xla*.

Di *default* gli *AddIn Excel* sono salvati nella directory *C:\Windows\ApplicationData\Microsoft \AddIns*. Se invece si vuole caricare l'*AddIn* automaticamente (allo *StartUp* di *Excel*) bisogna salvarlo nella directory *C:\WINDOWS\Application Data\Microsoft\Excel\XLSTART*.

IL CONTA CONTROLLI

Dopo questa lunga premessa descriviamo come implementare un *Addin Generico* che permette di contare e classificare i controlli presenti sui *Form* di un progetto *VB6*. L'*AddIn* oltre a contare e classificare i controlli permette di visualizzare, su un *ListBox*, gli *AddIn*, effettivamente, referenziati da *Visual Basic*. Inoltre faremo in modo che questo *AddIn*, dopo che è stato caricato, sia attivabile attraverso un pulsante, con icona, impostato sulla *Toolbar* dell'*IDE* di *Visual Basic*.
Per punti spieghiamo come implementare il tutto.

1. Create un nuovo progetto *AddIn*, come illustrato nel paragrafo *AddIn Generico*. Sull'oggetto *Connect* impostate i parametri come in *figura 2*. Nel progetto, inoltre, inserite un modulo *Bas* di

CARICARE E SCARICARE LE AGGIUNTE IN OFFICE

In Office 2000 o versioni successive per caricare le Aggiunte COM bisogna selezionare il menu Componenti Aggiuntivi COM. Questo menu, non essendo in primo piano, bisogna selezionarlo dalla maschera personalizza Menu. Descriviamo cosa fare: dal menu visualizza selezionate barra degli strumenti e poi Personalizza, sulla maschera che appare selezionate Strumenti/Componenti aggiuntivi COM, quindi trascinate la selezione sulla Toolbar dell'applicazione Office. Per caricare le Aggiunte non COM, invece, in Word dovete selezionare il menu Strumenti/modelli ed aggiunte, mentre in Excel dovete selezionare Strumenti/Componenti Aggiuntivi, ecc.

supporto e un *file di Risorse*, come mostrato in *figura 1*.

2. Nel *modulo Bas* inserite le seguenti dichiarazioni:

```
Public ArrayControlli(255) As String
Public ContContr As Long
```

Che servono, rispettivamente, per catalogare la descrizione dei controlli e il numero di controlli presenti sulla Form attivo. Nel *file di Risorse*, invece, inserite: l'icona dell'*AddIn* (che è un *Bitmap*) che comparirà sulla *Toolbar* dell'*IDE* e la relativa didascalia. Per capire come definire le risorse si controlli la *figura 4*.

3. Sulla *Form* dell'*AddIn* (nominata *FrmAddIn*) predisponete gli elementi che permettono di visualizzare il numero di controlli della *Form* attiva (una *Label* nominata *LabelCont*) e l'applicazione *Target* a cui l'*AddIn* è agganciato (una *Label* nominata *LabelList*). Inoltre predisponete i controlli per chiudere la *Form*, per visualizzare la lista dei tipi di controlli presenti sulla *Form* attiva e la lista degli *AddIns* installati nell'*IDE*; cioè tre *Command Button* (nominati *CancelButton*, *TipoControlli* e *MostraAddIn*) e un *ListBox* (nominato *List1*).

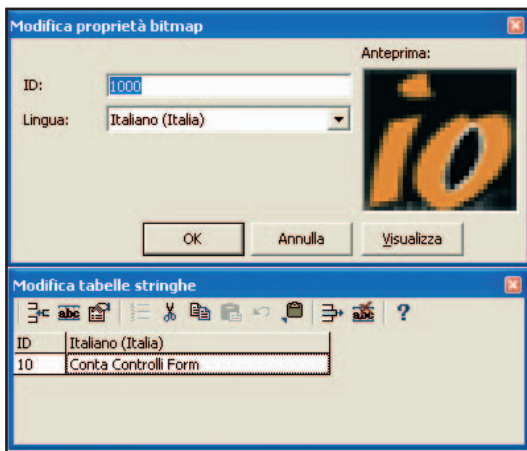



Fig. 4: **Le risorse impostate cioè un'immagine Bitmap ed un testo.**

4. Le dichiarazioni e il codice da prevedere nella FrmAddIn sono i seguenti.

```
Public VBInstance As VBIDE.VBE
'Istanza di Visual Basic
Public Connect As Connect
'Istanza del controllo Add In Designer

Private Sub CancelButton_Click()
'chiude la Form
Connect.Hide
```

```
End Sub

Private Sub MostraAddIn_Click()
'Visualizza gli Add In
Dim agg As Object
Dim i As Integer
List1.Clear
Labellist = " Aggiunte: " & VBInstance.FullName
i = 0
For Each agg In VBInstance.Addins
i = i + 1
List1.AddItem "Connesso: " +
 IIf(VBInstance.Addins.Item(i).Connect, "SI", "NO")
                                                  + _
 " -> " + VBInstance.Addins.Item(i).Description
Next
End Sub

Private Sub TipoControlli_Click()
'Visualizza i tipi di controllo
List1.Clear
Dim i As Integer
i = 1
While ArrayControlli(i) <> ""
List1.AddItem ArrayControlli(i)
i = i + 1
Wend
End Sub
```

La *MostraAddIn_Click* visualizza, nella *List1*, l'elenco degli *AddIn*, mentre nella *Labellist* presenta il nome dell'applicazione *Target* (cioè *VB6*). L'elenco degli *AddIn* è ricavato dalla collezione *VBInstance.Addins*. La *TipoControlli_Click*, semplicemente, visualizza sulla *List1* il contenuto dell'*ArrayControlli*. Quest'ultimo è impostato nella procedure *ContaControlli* contenute nel *Controllo Connect*.

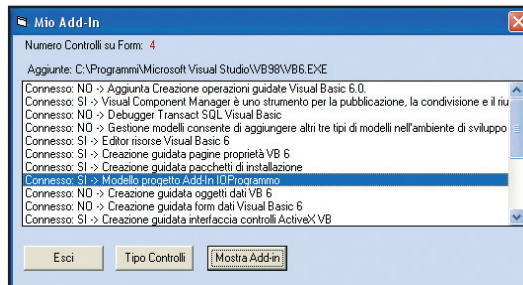
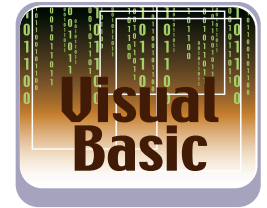
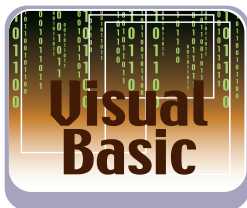



Fig. 5: **La form dell'esempio**

5. Per quanto riguarda l'AddInDesigner nei passi successivi, descriveremo soltanto il codice che abbiamo modificato o implementato ex novo, rispetto a quello prodotto da Visual Basic. Per evitare equivoci vi consigliamo di controllare il progetto completo contenuto nel CD allegato alla rivista.

6. Nella parte dichiarativa, dell'*AddInDesigner*, oltre alle dichiarazioni inserite da *Visual Basic*, dobbiamo far in modo di prevedere i seguenti elementi:

```
Public WithEvents CtlHandler  As VBControlsEvents
Dim mcmpCurrentForm As VBComponent
```

CtlHandle serve per gestire gli eventi dei controlli e per questo è del tipo *VBControlsEvents* invece *mcmpCurrentForm* conterrà il componente attivo.

7. Uno degli eventi fondamentali gestito dall'*AddInDesigner* è *AddinInstance_OnConnection*, che è generato quando l'*AddIn* è aggiunto all'*IDE* di *Visual Basic*. Nella *OnConnection* il parametro che decide come l'*AddIn* viene caricato è *ConnectMode* (si controllino le note inserite nel codice). Nel nostro esempio abbiamo modificato il codice dell'evento in modo che l'*AddIn* aggiunto possa essere attivato con un pulsante inserito sulla *ToolBar* dell'*IDE*. Per questo il codice da prevedere in *AddinInstance_OnConnection* è il seguente.

```
Private Sub AddinInstance_OnConnection(ByVal
                                          Application _
As Object, ByVal ConnectMode As
            AddInDesignerObjects.ext_ConnectMode, _
ByVal AddInInst As Object, custom() As Variant)

 On Error GoTo error_handler
 'Salva l'istanza di Visual Basic.
 Set VBInstance = Application
 If ConnectMode = ext_cm_External Then
  'L'aggiunta è stata avviata esternamente
  'da un altro programma o componente.
  Me.Show
 Else
  'L'aggiunta è stata avviata prima
  'della visualizzazione della finestra
  'di dialogo Apri progetto iniziale.
  'Set mcbMenuCommandBar =
    AddToAddInCommandBar("AddIn IOProgrammo")
  AddToCommandBar
  'Gestisce l'evento
  Set Me.MenuHandler =
   VBInstance.Events.CommandBarEvents(mcbMenu
                                    CommandBar)
 End If
 If ConnectMode = ext_cm_AfterStartup Then
  'L'aggiunta è stata avviata dopo la visualizzazione
  'della finestra di dialogo Apri progetto iniziale.
  If GetSetting(App.Title, "Settings",
           "DisplayOnConnect", "0") = "1" Then
   'Mostra il form collegato.
   Me.Show
  End If
 End If
 Exit Sub
error_handler:
 MsgBox Err.Description
End Sub
```

Quello che abbiamo modificato è la riga di codice seguente:

```
Set mcbMenuCommandBar =
                 AddToAddInCommandBar("AddIn
                              IOProgrammo")
```

Sostituita con

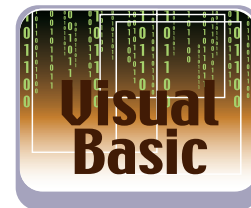
```
AddToCommandBar
```

La prima invoca la procedura che consente d'inserire il comando, che abilita l'*AddIn*, nel menu *Aggiunte*, la seconda, descritta sotto, consente d'inserire lo stesso comando come icona. Nel progetto d'esempio sono utilizzabili tutte e due.

8. La *OnConnection*, dunque, invoca la *AddToCommandBar* per aggiungere l'icona dell'*Add In* alla *Toolbar* di *Visual Basic*. Per questo nella *AddToCommandBar* sono caricate le risorse (icona e didascalia) impostate nel *file di risorse*.

```
Sub AddToCommandBar()
On Error GoTo AddToCommandBarErr
VBInstance.CommandBars(2).Visible = True
Set mcbMenuCommandBar =
  VBInstance.CommandBars(2).Controls.Add(1, , , _
VBInstance.CommandBars(2).Controls.Count)
mcbMenuCommandBar.Caption =
                              LoadResString(10)
Clipboard.SetData LoadResPicture(1000, 0)
mcbMenuCommandBar.PasteFace
Set MenuHandler =
   VBInstance.Events.CommandBarEvents(mcbMenu
```

ADDIN MANAGER E ADDIN TOOLBAR

Per caricare o scaricare gli AddIns si può usare l'AddIn Manager, selezionabile dal menu AddIns (Aggiunte). La finestra dell'AddIn Manager presenta l'elenco degli AddIns caricabili e le opzioni di caricamento, queste ultime sono selezionabili attraverso dei CheckBox. Le opzioni sono: Loaded/Unloaded per caricare o scaricare l'AddIn selezionato; Load On Startup per caricare allo Startup di Visual Basic; Command Line, per caricare, quando si fa lo StartUp di Visual Basic, da uno script o da un prompt di comando. Tra gli AddIns selezionabili c'è VB AddIn Toolbar: una Toolbar con la quale è possibile semplificare le operazioni di caricamento, attivazione e disattivazione degli AddIns. Questa è una ToolBar che inizialmente presenta solo un bottone (+/-) con il quali si richiama una maschera che permette di manipolare (cancellare o caricare) le DLL relative agli AddIns.

```
                                    CommandBar)
Exit Sub
AddToCommandBarErr:
MsgBox Err.Description
End Sub
```

Nella *AddToCommandBar*: si rende visibile la barra degli strumenti *Standard* e vi si aggiunge l'icona che permette d'invocare l'*AddIn*. In base a quello che abbiamo predisposto l'icona e la sua didascalia sono recuperati dal *file di risorse*. Facciamo notare che l'icona è aggiunta sulla barra degli *strumenti Standard* a destra dei pulsanti e che essa è prima copiata negli appunti e poi impostata nel pulsante. Infine è impostato l'evento ad essa collegato, cioè *MenuHandler_Click*.

9. Quando si clicca l'icona associata all'*AddIn* è generato l'evento *MenuHandler_Click* che

Fig. 6: **L'icona che avvia l'AddIn.**

invoca la procedura *Show()* che mostra la Form *frmAddIn* (cioè *mfrmAddIn*). Il codice di questo evento e della *Show* sono descritti sotto.

```
Private Sub MenuHandler_Click(ByVal
                              CommandBarControl _
As Object, handled As Boolean,
                              CancelDefault As Boolean)
Show
End Sub
Sub Show()
If mfrmAddIn Is Nothing Then
Set mfrmAddIn = New frmAddIn
End If
Set mfrmAddIn.VBInstance = VBInstance
Set mfrmAddIn.Connect = Me
FormDisplayed = True
mfrmAddIn.Show
ContaControlli
mfrmAddIn.LabelCont = ContContr
'.....modifiche
End Sub
```

9. La *Show* tra l'altro invoca la procedura che conta i controlli cioè la *ContaControlli* e aggiorna la *Label* presente sulla *frmAddIn*.

```
Public Sub ContaControlli()
```

```
On Error GoTo ContaControlliErr
Dim ctl As VBControl
If VBInstance.ActiveVBProject Is Nothing Then Exit
                                              Sub
Set mcmpCurrentForm =
                  VBInstance.SelectedVBComponent
'Verifica se il componente è valido
If mcmpCurrentForm Is Nothing Then
Exit Sub
End If
ContContr = 0
For Each ctl In
              mcmpCurrentForm.Designer.VBControls
ContContr = ContContr + 1
ArrayControlli(ContContr) = ctl.ClassName
Next
Exit Sub
ContaControlliErr:
MsgBox Err.Description
End Sub
```

S'intuisce che la *ContaControlli* andrebbe migliorata, prevedendo per esempio la gestione degli *Array* di controlli e la catalogazione e sommatoria dei controlli dello stesso tipo.

10. Infine vediamo gli eventi che sono generati, quando s'inserisce o cancella un controllo dalla *Form* attiva, cioè *CtlHandler_ItemAdded* e *CtlHandler_ItemRemoved*.

```
Private Sub CtlHandler_ItemAdded(ByVal
                    VBControl As VBIDE.VBControl)
`aggiunto un controllo
ContContr = ContContr + 1
mfrmAddIn.LabelCont = ContContr
End Sub
Private Sub CtlHandler_ItemRemoved(ByVal
                    VBControl As VBIDE.VBControl)
`eliminato un controllo
ContContr = ContContr - 1
mfrmAddIn.LabelCont = ContContr
End Sub
```

Anche in questo caso notate che mancano le istruzioni che permettono di aggiornare l'*ArrayControlli* che contiene la descrizione degli oggetti.

CONCLUSIONE

La base teorica su cui si poggiano gli Addin è leggermente complessa, ma se proverete ad eseguire l'esempio scoprirete che tutto è molto più semplice di quanto non sembri.

Massimo Autiero

SVILUPPIAMO UN WIZARD IN C#

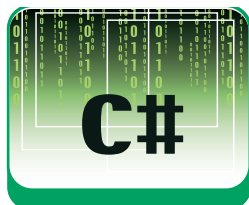
IL .NET FRAMEWORK 2.0 NON DISPONE DI UN CONTROLLO WIZARD. IN QUESTO ARTICOLO NE CREEREMO UNO E LO UTILIZZEREMO PER REALIZZARE UNA PROCEDURA AUTOMATICA CHE AIUTI L'UTENTE AD EFFETTUARE UN UPLOAD VIA FTP

REQUISITI

Conoscenze richieste
Conoscenze di programmazione in C#

Software
.NET Framework 2.0, Visual Studio 2005

Impegno

Tempo di realizzazione

Le procedure guidate, anche chiamate wizard, sono quelle particolari applicazioni che guidano l'utente in maniera facilita nella realizzazione di un'azione. Windows utilizza questa tecnica in diversi scenari. I più classici esempi di wizard sono rappresentati dalle procedure di installazione. Un wizard è, in poche parole, un'applicazione o una parte di essa che guida l'utente nell'esecuzione di una serie di step tra loro collegati. Un wizard è, ad esempio, la registrazione di un utente su un sito internet, la creazione di un sito web su IIS.
Con la versione 2.0 del .NET Framework anche ASP.NET fornisce un controllo wizard che può essere riutilizzato a proprio piacimento in qualsiasi sito web. Di contro, però, non esiste un controllo simile per le applicazioni desktop. Proveremo, in questo articolo, a sviluppare un controllo che ci consenta, in pochi semplici passi, di realizzare un wizard anche complesso in grado di essere riutilizzato ed adattato alle specifiche esigenze delle nostre varie applicazioni. Nella pratica svilupperemo un wizard in grado di guidare l'utente nell'invio di un file ad un percorso ftp.

SVILUPPIAMO IL CONTROLLO

La soluzione meno efficiente per realizzare il nostro wizard è quella di sviluppare una nuova form per ogni passo da compiere.

Figura 1 – Lo schema delle classi del controllo Wizard.

Ovviamente si tratta di una soluzione veramente poco raffinata. La nostra idea invece è quella di realizzare una sola form, al cui interno inseriremo un controllo che conterrà di volta in volta gli elementi da visualizzare, che saranno dinamici. In sostanza al clic sul bottone Next per esempio verrà cancellato il contenuto attuale del pannello e sostituito con quello necessario alla realizzazione del passo corrente.
Prima di incominciare facciamo un po' l'inventario di quello che ci occorre: innanzitutto ci serve un form, chiamato *WizardForm*,. Un altro elemento di cui avremo sicuramente è uno usercontrol, definito *WizardPanelBase*, che costituirà il pannello base all'interno de quale andremo poi a posizionare gli elementi necessari a realizzare i vari step del wizard. Il controllo stesso, così come i suoi controlli figlio, saranno da qui definiti come panels. Creeremo quindi uno *user control* base che si occuperà di gestire le caratteristiche comuni a tutti i panels.
Di conseguenza ogni singolo panel dovrà ereditare dal panel base per poter essere aggiunto alla collezione del wizard.
In Visual Studio 2005 creiamo un nuovo progetto Windows Class Library che chiameremo WizardControl. Contemporaneamente creiamo una nuova soluzione che chiameremo WindowsWizardApplication.
Nel progetto *WizardControl* creiamo innanzittutto il controllo WizardPanelBase che utilizzeremo come base per l'implementazione dei panels del wizard. Non esamineremo ora il codice che compone lo *user control*, ma ci limiteremo semplicemente ad impostare la dimensione da applicare. Per la realizzazione del nostro esempio abbiamo utilizzato una lunghezza pari a 432 pixel ed una altezza pari a 329 pixel.
Per il completamento delle prime operazioni dobbiamo ora creare il form che costituirà il contenitore dei nostri panels. Aggiungiamo

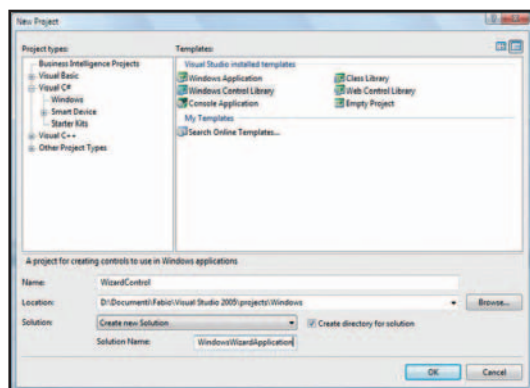



Figura 2 – La creazione della soluzione.

quindi al progetto il form *WizardForm* e, utilizzando il designer di Visual Studio, lo componiamo con un controllo picturebox sulla sinistra, il controllo *WizardPanelBase*, da poco creato, al centro e tre controlli button sulla base del form, fino ad ottenere quanto visualizzato in figura 3.
Nel *WizardForm* aggiungiamo una proprietà che, utilizzando la collection generica List<T>, gestisce l'elenco dei diversi panels che saranno utilizzati dal wizard:
Ottenuta la lista dei vari panels, dobbiamo ora decidere quale, tra quelli impostati, è il panel attivo. Per questo scopo utilizzeremo un metodo *SetCurrent Panel* che sulla base dell'indice passato come parametro attiva il nuovo panel e disattiva quello precedente. La funzione utilizza anche un campo privato *_activePanel* per mantenere, tra le diverse chiamate, un riferimento al panel attualmente attivo:
La funzione *SetCurrentPanel*, sopra riportata, recupera l'istanza del panel da attivare, in base all'indice passato come parametro, dalla collezione *_panels*. Successivamente disattiva il panel attivo ed attiva il nuovo panel modificando la proprietà visible. Inoltre, se non presente, il novo panel viene aggiunto alla collezione dei controlli del panel container *WizardPanelContainer*.
La funzione *SetButtons*, richiamata alla fine del codice, imposta la visualizzazione dei bottoni in base al pannello visualizzato. Per motivi di spazio non riportiamo il codice completo, che è comunque disponibile nel cd allegato alla rivista.
Nel gestore dell'evento *Load* del form *WizardForm*, aggiungiamo la chiamata alla funzione *SetCurrentPanel* per eseguire l'impostazione al primo caricamento del form. Inoltre aggiungiamo una chiamata alla funzione *SetCurrentPanel* nei gestori degli eventi clic dei pulsanti *Next* e *Previous* per consentire la navigazione tra i panels:
Dopo aver completato questo passaggio possiamo ora utilizzare il nostro controllo.

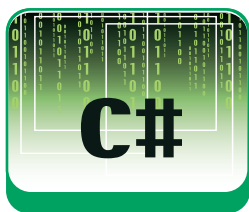
Figura 3 – Il WizardForm all'interno del designer di Visual Studio 2005.

UTILIZZIAMO IL CONTROLLO

Per vedere all'opera il controllo appena sviluppato dobbiamo creare, ed inserire nella nostra soluzione, un nuovo progetto *Windows Application* chiamandolo *FTPUpload Wizard*. Il primo passo è quello di aggiungere un riferimento al progetto *Wizard Control*. Facciamo ora clic con il tasto destro sul nome del progetto e selezioniamo *Add Reference* o *Aggiungi Riferimento* per chi possiede la versione italiana di Visual Studio 2005. Nella finestra di dialogo scegliamo la finestra *Projects*, selezioniamo *WizardControl* e facciamo clic sul tasto *OK*. Una volta aggiunto il riferimento possiamo iniziare a creare i nostri panels. Per la realizzazione della nostra applicazione abbiamo bisogno di almeno quattro panels:

- **WelcomePanel**: è il primo pannello che visualizziamo e che riporta il messaggio di benvenuto;
- **SelectFilePanel**: qui selezioniamo il file ed

GLI EVENTI PANELLOAD E PANELUNLOAD

Nell'articolo, per comodità e per motivi di spazio, abbiamo utilizzato l'evento *VisibileChanged*. Questo evento è disponibile solo con la versione 2.0 del .NET Framework. Qualora volessimo realizzare la stessa applicazione per una delle precedenti versioni, dovremmo implementarci degli eventi personalizzati da richiamare prima della disattivazione del panel per l'unload, e dopo l'attivazione del panel per simulare l'evento load. Questo a differenza del normale evento load, ci consentirà di gestire situazioni in cui abbiamo bisogno di effettuare operazioni particolari.

impostiamo le proprietà del server ftp su cui dobbiamo poi effettuare l'upload;

- **SendFilePanel**: in questo pannello effettuiamo l'upload e controlliamo lo stato di avanzamento dell'operazione;
- **FinishPanel**: nell'ultimo pannello visualizziamo il messaggio che ci comunica l'esito dell'operazione;

Per creare i panels sopra elencati dobbiamo seguire alcuni semplici passi. Facciamo clic con il tasto destro del mouse sul nome del progetto *FtpUploadWizard* e selezioniamo *Add -> New Item*. Qui scegliamo il template *Inherited User Control*, nominiamo il controllo come *WelcomePanel.cs* e clicchiamo su *OK*. Visual Studio 2005 ci chiede da quale controllo vogliamo ereditare. Nella finestra di dialogo scegliamo il controllo *WizardPanelBase* e clicchiamo su *OK*.
Ripetiamo la stessa operazione per gli altri tre controlli sopra indicati creando così i vari panels del nostro wizard. Ora apriamo il file *Program.cs* ed inseriamo questo codice nel metodo *Main*:
questo consente l'inserimento dei vari controlli panel nella collection e l'avvio del wizard utilizzando l'*Application.Run*. Ora siamo in grado di eseguire il wizard e di scorrere tra i diversi panels. Vedremo ora come gestire ogni singolo panel e consentire la condivisione delle informazioni raccolte. Prima di passare all'implementazione dei singoli panels, infatti, dobbiamo gestire il contesto del wizard.

Figura 4 – La finestra di dialogo per la selezione dello usercontrol base.

LA GESTIONE DEL CONTESTO

Una soluzione, quella poi implementata in questo articolo, per gestire il contesto è la creazione di una proprietà *Context* di tipo *Dictionary<String, Object>* nel *WizardForm* del progetto *WizardControl*:
Questa proprietà è facilmente accessibile dai controlli che ereditano da *WizardPanelBase*. In quest'ultimo, infatti, aggiungiamo una proprietà che accede alla proprietà *Context* del form *WizardForm*, accessibile utilizzando la proprietà *ParentForm* comune a tutti gli usercontrols, e rendendone di fatto trasparente l'utilizzo e la condivisione a livello di singolo wizard:
In questo modo recuperiamo un riferimento al form contenitore e qui effettuiamo un cast per mantenerlo ed evitare di rifarlo in seguito.

IMPLEMENTIAMO IL WIZARD

Dopo aver implementato la gestione del contesto, possiamo ora realizzare i vari panels che realizzano il wizard. Nel progetto *FtpUpload Wizard* apriamo il controllo *WelcomePanel* ed aggiungiamo una semplice label dove visualizziamo il nostro messaggio di benvenuto. La stessa cosa la possiamo fare anche *Finish Panel* dove inseriamo il messaggio di chiusura. Ora apriamo il *SelectFilesPanel* ed aggiungiamo due controlli groupbox. Nel primo inseriamo un controllo textbox ed un controllo button che ci consentiranno di selezionare il file da inviare al server ftp.
Nel secondo group box, invece, inseriamo i dati per l'identificazione del percorso ftp, la porta da utilizzare ed eventualmente le informazioni di autenticazione necessarie. Il risultato dovrebbe essere simile a quello visualizzato in figura 5.

GESTIAMO I PULSANTI

Una possibile miglioria che possiamo apportare al nostro wizard è la gestione dei buttons. Questa potrebbe avvenire perché magari necessitiamo di abilitare o disabilitare i buttons a seconda delle operazioni che stiamo effettuando. Riallacciandoci all'esempio dell'articolo, potremmo disabilitare il button *Next* nel momento in cui stiamo effettuando l'upload. Una delle possibili soluzioni potrebbe essere quella di rendere pubblici i buttons e controllarli dai singoli panels, ma così facendo creeremo un legame troppo forte. Diverso è se decidiamo di passare una variabile di stato. Potremmo ad esempio avere un enum così composto:

in questo modo potremmo sfruttare la modalità detta bitwise per impostare lo stato dei buttons. Nel *WizardForm* aggiungiamo un metodo che ci consente di modificare la proprietà *Enabled*:

a questo punto possiamo accedere al metodo e passare il relativo stato. Se volessimo, ad esempio, disabilitare l'uso dei buttons dal *SendFilePanel* durante l'invio per poi riabilitarli alla fine potremmo scrivere:

Ed il gioco è fatto. Il vantaggio nell'uso di questa tecnica è un accoppiamento molto limitato tra il WizardForm e i diversi panels.

Figura 5 – Il panel SelectFilePanel nel designer di Visual Studio 2005.

Dopo aver raccolto i dati dobbiamo fare in modo di renderli disponibili agli altri controlli del wizard sfruttando, appunto, il contesto. Nella procedura di gestione *SetCurrentPanel* del *WizardForm* abbiamo utilizzato la proprietà *visible* per attivare o disattivare i controlli. Il framework in versione 2.0 ci mette a disposizione un nuovo evento generato dalla modifica su tale proprietà. Semplicemente sottoscrivendo l'evento possiamo intercettare il momento in cui carichiamo o scarichiamo il pannello e quindi effettuare le operazioni necessarie. Nella gestione dell'evento *Visible Changed* impostiamo le variabili di contesto che saranno poi riutilizzate dal panel successivo:

Ora dobbiamo preparare il pannello che eseguirà l'invio dei dati. Apriamo lo usercontrol *SendFilePanel.cs* e nel designer aggiungiamo una label, chiamata *lblTitle* che visualizzerà lo stato dell'operazione, un button (*btnSend*) che effettuerà l'invio e una progressbar (*progress*) che ci aggiornerà sull'avanzamento della procedura di invio. Lo usercontrol sarà simile a quello visualizzato in figura 6.

Come prima cosa sottoscriviamo l'evento

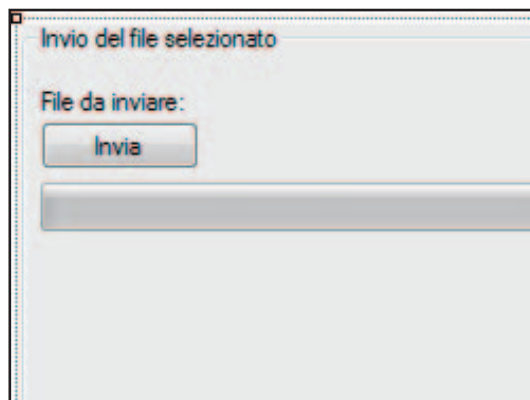



Figura 6 – Il SendFilePanel all'interno del designer di Visual Studio 2005.

VisibleChanged, così come fatto per lo usercontrol *SelectFilePanel.cs*, e recuperiamo le informazioni di contesto che utilizzeremo per l'invio:

Associamo poi all'evento clic del button il codice per l'esecuzione dell'invio vero e proprio:

la procedura effettua l'invio aprendo lo stream del file e scrivendolo direttamente nello stream di richiesta. Le impostazioni recuperate consentono di connettersi al server, eventualmente procedere con l'autenticazione dell'utente, e infine di effettuare l'invio. Al termine della procedura viene visualizzato un messaggio di conferma della buona riuscita della procedura di invio. Se proviamo ad eseguire l'applicazione potremo apprezzarne il risultato.

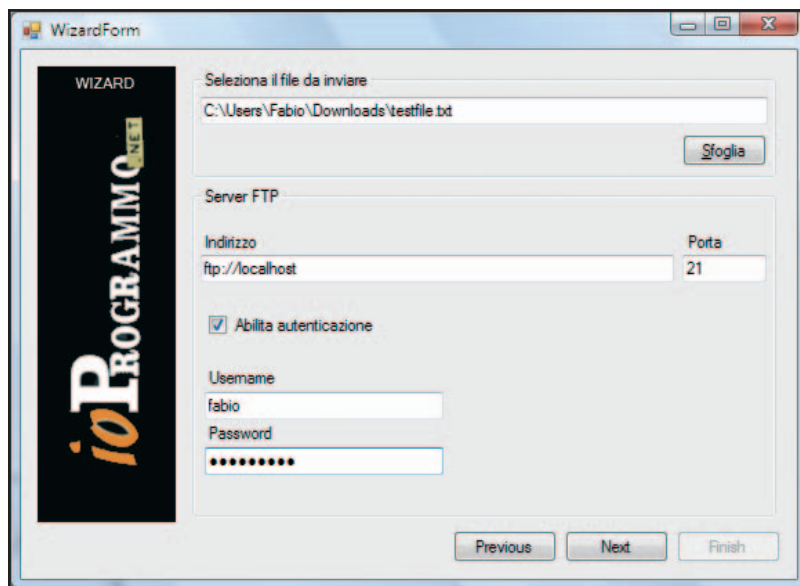
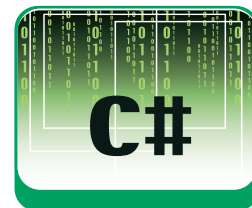



Figura 7 – Il wizard in esecuzione.

CONCLUSIONI

In questo articolo abbiamo visto come realizzare un semplice wizard adattabile a qualsiasi situazione perché sviluppato come controllo riutilizzabile. Sfruttando alcune caratteristiche del .NET Framework 2, abbiamo implementato un wizard in grado di inviare un file, selezionato sul disco locale, verso un server ftp. Alcune migliorie al controllo immediatamente attuabili potete trovarle nei box di approfondimento all'articolo. Per qualsiasi domanda, chiarimento o dubbio potete scrivere sul forum di ioProgrammo

Fabio Cozzolino

L'AUTORE

Fabio Cozzolino sviluppa software in da circa 10 anni. Si è appassionato al .NET Framework fin dalla prima versione sviluppando principalmente applicazioni web, raggiungendo la certificazione MCAD. Negli ultimi due anni si è dedicato alla produzione di software principalmente in architetture distribuite e service-oriented. È cofondatore dello user group DotNetSide con il quale organizza eventi periodici dedicati al mondo .NET.

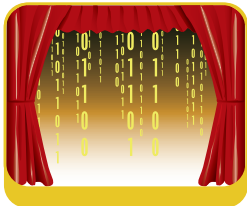
VISUAL BASIC 9.0 QUASI PRONTO!

TI SVELIAMO IN ANTEPRIMA TUTTE LE NOVITÀ CHE CARATTERIZZERANNO LA NUOVA VERSIONE DI UNO DEI LINGUAGGI PIÙ USATI AL MONDO. VEDREMO CHE L'INTRODUZIONE DEL FRAMEWORK .NET 3.0 PORTA CON SE ALCUNE INNOVAZIONI ENTUSIASMANTI

Conoscenze richieste
conoscenza di base di VB.NET

Software
Visual Studio 2005 e in versione Express

Impegno

Tempo di realizzazione

Nelle cucine di casa Microsoft sta nascendo il nuovo Visual Basic. Per la precisione si tratta della versione 9.0 del notissimo linguaggio di programmazione (dal nome in codice "Orcas"). Le novità sono molte e "succulente" e tutte orientate a semplificare ulteriormente la vita a chi utilizza un modello di programmazione orientato agli oggetti ed all'integrazione con il nuovo framework LINQ.

Alcune delle nuove caratteristiche saranno:

- Variabili implicitamente tipizzate
- Query integrate
- Inizializzatori di Oggetti e Collections
- Tipi Anonimi
- Supporto integrato a XML
- Binding dei delegati
- Tipi Nullable
- Interfacce dinamiche

UN "GIRO DI PROVA"

Per il nostro primo "giro di prova" delle nuove caratteristiche ci rifacciamo all'anteprima del linguaggio pubblicata da Microsoft. Quindi riprendiamo l'esempio in essa proposto che prende ad esempio una classe basata sulla raccolta di informazioni geo-politiche sugli Stati rappresentati con:

```
Class Country
  Public Property Name As String
  Public Property Area As Float
  Public Property Population As Integer
End Class
```

Già qualche differenza l'avete notata? Ebbene sì, intanto le proprietà si potranno dichiarare anche senza Get e Set, nel VB attuale ogni proprietà sarebbe dovuta essere dichiarata come:

```
    Private _Nome As String
    Public Property Nome() As String
        Get
            Return _Nome
        End Get
        Set(ByVal Value As String)
            _Nome = Value
        End Set
    End Property
```

Insomma, un bel risparmio di caratteri da digitare sulla tastiera no? Ma il bello deve ancora venire: come faccio ad ottenere una collection di oggetti? Semplicemente con:

```
Dim Countries = _
  { New Country{ _
      Name := "Palau", Area := 458, Population :=
                                          16952 }, _
    New Country{ _
      Name := "Monaco", Area := 1.9, Population
:= 31719 }, _
    New Country{ _
      Name := "Belize", Area := 22960, Population
:= 219296 }, _
    New Country{ _
      Name := "Madagascar", Area := 587040,
                        Population := 13670507 } _
  }
```

PROVA SU STRADA

È possibile provare "su strada" il nuovo Visual Basic anche senza aspettare la nuova versione di Visual Studio. Microsoft mette a disposizione un'estensione a Visual Studio chiamata Linq Preview e disponibile sul sito http://msdn.microsoft.com/vbasic/future/ dove è possibile trovare tutta la documentazione sul nuovo Visual Basic e sulla tecnologia LINQ. L'estensione funziona solo con la versione inglese di Visual Studio, chi avesse Visual Studio in italiano può effettuare i test installando gratuitamente Visual Basic Express in inglese nel sito http://msdn.microsoft.com/vstudio/express/vb/download/ . L'estensione introduce nell'IDE tre nuovi template di progetti: LINQ Windows Application, LINQ Console Application, e LINQ Class Library che possono essere utilizzati per iniziare a familiarizzare con il linguaggio (naturalmente ancora non è consigliabile utilizzare VB9 in produzione).

Abbiamo così ottenuto, al volo, una lista, che possiamo interrogare con il meccanismo di query integrato nel linguaggio.
Se, ad esempio, vogliamo ottenere tutti i paesi con popolazione inferiore ad un milione di abitanti potremo semplicemente dichiarare:

```
Dim SmallCountries = From Country In Countries _
        Where Country.Population < 1000000 _
Select Country
For Each Country As Country
                              In SmallCountries
  Console.WriteLine(Country.Name)
Next
```

Per chi, come chi scrive, ha passato lunghe ore a creare *collection* di oggetti con i relativi metodi di filtro questa sintassi così concisa e pulita ha dell'incredibile! Ma torniamo sul nostro esempio per scoprire qualche particolare in più.
In primo luogo sulla dichiarazione della variabile *Countries* :

```
Dim Countries = _
  { New Country { Name := "Palau", Area := 458,
                          Population := 16952 }, _
  … _
  }
```

che utilizza la nuova sintassi per l'**inizializzazione degli oggetti** per creare un nuovo oggetto complesso con una sintassi simile a quella della dichiarazione *With*. Da notare anche che *Countries* non dichiara esplicitamente un tipo (List o Array) ma utilizza il concetto, tutto nuovo, di **variabile implicitamente tipizzata**. Il compilatore, cioè, desume il tipo della variabile dal contenuto ad essa assegnato. L'estrazione dei dati utilizza una sintassi simile a SQL per il filtro dei dati, si tratta del nuovo concetto di query integrate nel linguaggio:

```
Dim SmallCountries = From Country
                                   In Countries _
        Where Country.Population < 1000000 _
Select Country
```

anche qui abbiamo un esempio di variabile implicitamente tipizzata in quanto il compilatore desume dal contesto che *SmallCountries* sia un tipo IEnumerable(Of Country). In questo primo, semplice esempio abbiamo visto un buon numero delle nuove caratteristiche del prossimo Visual Basic.
Scendiamo maggiormente nel dettaglio analizzanzo approfonditamente tutte le singole novità. Scopriremo alcune cose utili.

VARIABILI IMPLICITAMENTE TIPIZZATE

Una variabile implicitamente tipizzata è quella in cui il tipo viene dedotto, dal compilatore, dall'espressione di inizializzazione (ciò che c'è dal lato destro del segno "=", per intendersi). Così:

```
Dim Population = 31719
Dim Name = "Belize"
Dim Area = 1.9
Dim Country = New Country{
                        Name := "Palau", …}
```

È lo stesso che scrivere, in modo esplicito :

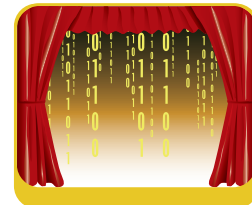
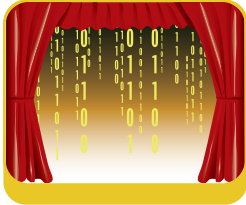
```
Dim Population As Integer = 31719
Dim Name As String = "Belize"
Dim Area As Float = 1.9
Dim Country As Country = New Country{
                Name :=  "Palau", …}
```

A molti questo sistema di scrittura può far venire in mente i linguaggi di scripting come VBScript o Javascript dove non esistono affatto i tipi o il Late Binding, l'associazione posticipata al tipo, usata anche nel Visual Basic.
Ma non è così, in realtà l'associazione viene qui fatta automaticamente dal compilatore, non in fase di esecuzione, per cui la variabile, a runtime, si trova perfettamente tipizzata.
Questo è anche il motivo per cui le variabili implicitamente tipizzate vengono supportate anche con l'opzione *Option Strict*.
Mentre il *Late Binding* è ancora possibile ma dev'essere esplicitamente dichiarato con l'assegnazione del tipo *Object* alla variabile:

```
Dim Country As Object = new Country{ Name :=
                                "Palau", … }
```

Anche i cicli *For…Next e For Each…Next* beneficiano, come sinteticità dell'espressione, dal nuovo concetto di **variabili implicitamente tipizzate**. Per cui, nel ciclo:

```
For Each Dim Country In SmallCountries
  Console.WriteLine(Country.Name)
Next
```

Il tipo della variabile *Country* sarà implicitamente ricavato dal tipo degli elementi che compongono l'insieme *SmallCountries*. Si tratta sicuramente di una delle novità che maggiormente caratterizzano il nuovo ambiente e che consentono di ottenere un notevole risparmio di tempo.

INIZIALIZZATORI DI OGGETTI E COLLECTION

Nel nuovo Visual Basic, l'abbiamo già visto, ci sarà la possibilità di inizializzare (cioè creare un istanza) un oggetto automaticamente.
Come, ad esempio in :

```
Dim Palau = New Country { _
  Name := "Palau", _
  Area := 458, _
  Population := 16952
}
```

Dove vengono assegnati i valori delle proprietà direttamente nel costruttore (si noti che la classe non disponeva di un costruttore *New* con tali parametri).
In pratica si elimina la necessità di fornire più costruttori in fase di progettazione della classe (la cosa naturalmente resta comunque ancora possibile).
La nuova tecnica di inizializzazione si applica anche alle collection che supportino il metodo Add per l'aggiunta di un nuovo elemento.
Prendiamo ad esempio un'altra classe:

```
Public Class City
  Public Property Name As String
  Public Property Country As String
  Public Property Longitude As Float
  Public Property Latitude As Float
End Class
```

possiamo creare, agevolmente, una nuova lista di City con:

```
Dim Capitals = New List(Of City){ _
  { Name := "Antanarivo", _
    Country := "Madagascar", _
    Longitude := 47.4, _
    Lattitude := -18.6 }, _

  { Name := "Belmopan", _
    Country := "Belize", _
    Longitude := -88.5, _
    Latitude := 17.1 }, _

  { Name := "Monaco", _
    Country := "Monaco", _
    Longitude := 7.2, _
    Latitude := 43.7 }, _

  { Country := "Palau",
    Name := "Koror", _
    Longitude := 135, _
    Latitude := 8 } _
}
```

TIPI ANONIMI

A volte, filtrando un'insieme, è necessario estrarre solo alcune proprietà dagli oggetti che compongono la collection. Ad esempio, dalla lista precedente potremmo voler ricavare il nome della città e dello Stato di tutte le capitali che si trovano tra il Tropico del Cancro ed il Tropico del Capricorno.
Ci servirebbe quindi un tipo per "ospitare" i risultati composto solo dalle proprietà *Name* e *Country* visto che *Longitude* e *Latitude* sono superflue.
In Visual Basic 9 questo tipo "di servizio" viene automaticamente creato con:

```
Const TropicOfCancer = 23.5
Const TropicOfCapricorn = -23.5
Dim Tropical = From City In Capitals _
                    Where TropicOfCancer <=
                         City.Latitude _
                  AndAlso City.Latitude >=
                              TropicOfCapricorn _
                    Select New {Name :=
             City.Name, Country := City.Country}
```

la variabile *Tropical* sarà implicitamente tipizzata come

```
IEnumerable(Of { Name As String,
                     Country As String })
```

e gli oggetti che compongono l'insieme apparterranno ad un tipo creato dal sistema, nello specifico:

```
Class _Name_As_String_Country_As_String_
    Public Property Name As String
    Public Property Country As String
    …
End Class
```

Nell'ambito dello stesso programma, il compilatore quando troverà più tipi anonimi uguali provvederà automaticamente a fonderli in un unico tipo.
Inoltre, poiché i tipi anonimi saranno spesso utilizzati come dei sottoinsiemi di proprietà di tipi esistenti, Visual Basic 9 consente anche la notazione abbreviata:

```
New { City.Name, City.Country }
```

in luogo di

```
New { Name := City.Name, Country :=
City.Country }
```

Quindi l'espressione di Select si potrà scrivere

anche come:

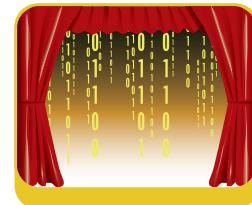
```
Dim Tropical = From City In Capitals _
            Where TropicOfCancer
                        <= City.Latitude _
        AndAlso City.Latitude >=
                        TropicOfCapricorn _
            Select City.Name, City.Country
```

SUPPORTO INTEGRATO A XML

L'integrazione con XML fa un ulteriore passo avanti con il supporto all'API XLinq, integrata nel nuovo Framework.
La costruzione di XML sarà integrata direttamente nel linguaggio con XML Literals che permette di inserire codice XML frammisto a codice Visual Basic attraverso una sintassi ASP-Like. Vediamo, ad esempio, il modo di estrarre semplicemente le informazioni di una query su una Collection in formato XML:

```
Dim CountriesWithCapital As XElement = _
    <Countries>
    <%= From Country In Countries, City
                                In Capitals _
        Where Country.Name = City.Country _
    Select <Country Name=<%= Country.Name %>
                Density=
                <%= Country.Population /
                        Country.Area %>>
            <Capital>
             <Name>
                <%= City.Name %>
            </Name>
             <Longitude>
                <%= City.Longitude %>
             </Longitude>
             <Latitude><%= City.Latitude
                                %></Latitude>
            </Capital>
          </Country> _
        %>
    </Countries>
```

e guardate che tutto questo lo faremo dentro Visual Basic! I marcatori <%= %> proprio come in ASP indicano al compilatore la presenza di codice.
Il codice precedente produce un XML come questo:

```
<Countries>
 <Country Name="Palau"
            Density="0.037117903930131008">
  <Capital>
<Name>Koror</Name>
            <Longitude>135</Longitude>
            <Latitude>8</Latitude>
  </Capital>
 </Country>
 <Country Name="Monaco"
            Density="16694.21052631579">
  <Capital>
<Name>Monaco</Name>
        <Longitude>7.2</Longitude>
        <Latitude>3.7</Latitude>
  </Capital>
 </Country>
 <Country Name="Belize"
            Density="9.5512195121951216">
  <Capital>
   <Name>Belmopan</Name><Longitude>
    88.5</Longitude><Latitude>17.1</Latitude>
  </Capital>
 </Country>
 <Country Name="Madagascar"
            Density="23.287181452711909">
  <Capital>
   <Name>Antananarivo</Name>
   <Longitude>47.4</Longitude><Latitude>-
                        18.6</Latitude>
  </Capital>
  </Country>
</Countries>
```

Oltre alla costruzione dinamica di XML Visual Basic 9 permette anche il Late Binding sull'XML, cioè potremo interrogare un oggetto XML con sintassi analoga a XPath.
Se prendiamo, ad esempio, la variabile CountriesWithCapital , che abbiamo prima valorizzato, potremmo interrogarla semplicemente con:

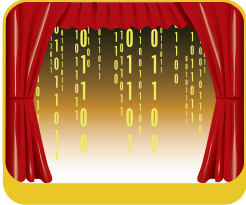
```
For Each Dim Country In
            CountriesWithCapital.Country
            Console.WriteLine("Density = "
                    & Country.@Density)
   Console.WriteLine("Latitude = " &
                Country...<Latitude>(0))
Next
```

In fase di Runtime il codice viene tradotto in questo modo:

```
CountriesWithCapital.<Country>
```

diventa

```
CountriesWithCapital.Elements("Country"),
```

ovvero un insieme di elementi XML chiamati "Country"

```
Country.@Density
```

viene tradotto in

```
Country.Attribute("Density")
```

ovvero l'attributo "Density" del nodo "Country" L'espressione

```
Country...<Latitude>(0)
```

viene tradotta nella combinazione di elementi

```
ElementAt(Country.Descendants(Latitude),0)
```

0ovvero nel primo elemento "Latitude" contenuto nel nodo "Country".

QUERY INTEGRATE

Il prossimo Visual Basic permetterà di effettuare selezioni in insiemi di oggetti con una sintassi del tutto (e volutamente) simile all'SQL. Sarà quindi possibile filtrare gli oggetti con espressioni From…Where…Select… proprio come per le tabelle di un database:

```
Dim SmallCountries = From Country In Countries _
            Where Country.Population < 1000000 _
Select Country
```

È anche possibile utilizzare il nuovo operatore It per riferirsi all'oggetto che costituisce la riga corrente come in:

```
Dim CountriesWithCapital = _
            From Country In Countries, City
                                        In Capitals _
            Where Country.Name = City.Country _
            Select It
```

Le similitudini con SQL si estendono anche al sort dei record, effettuato con la familiare sintassi *Order By*:

```
Dim Sorted = From Country In Countries, City In
                                        Capitals _
             Where Country.Name = City.Country _
             Order By City.Longtitude Asc,
                      Country.Population Desc _
            Select Country.Name
```

Non mancano nemmeno gli operatori di aggregazione come Min, Max, Count, Avg, Sum ecc… Per ottenere il conteggio di potrà scrivere semplicemente un codice con una sintassi di questo tipo:

```
Dim N As Integer = _
From Country In Countries _
Where Country.Population < 1000000 _
Select Count(Country)
```

Oppure prevedere aggregazioni più complesse come :

```
Dim R As { Total As Integer, Density
                                 As Double } = _
  From Country In Countries _
  Where Country.Population < 1000000
  Select New { Total := Count(Country), _
               Density :=
                Avg(Country.Population/Country.Area) }
```

TIPI "NULLABLE"

La discrasia tra il valore NULL presente nei database relazionali con i tipi di .NET era già stata affrontata dalla versione attuale prevedendo il tipo generico Nullable(Of T As Structure) che si applica ai tipi struttura come Integer, String, Boolean ecc…
In nuovo Visual Basic semplifica ulteriormente la dichiarazione di tipi Nullable utilizzando il suffisso "?" nella dichiarazione del tipo, come in:

```
Property NullableDate As Date?
Property NullableN As Integer?
```

Le operazioni che coinvolgono un oggetto dal valore Nothing restituiranno, naturalmente, sempre Nothing, come ad esempio in:

```
Dim NullableN As Integer?
NullableN = Nothing
Dim Result As Integer? = NullableN + 3
```

Result sarà appunto Nothing.

BINDING DEI DELEGATI

Attualmente quando si crea un *delegato* attraverso gli operatori AddressOf o Handles i metodi collegati a quel *delegato* devono avere gli stessi parametri. Ad esempio, l'evento OnClick di un controllo Button deve essere conforme al delegato :

```
Delegate Sub EventHandler(sender As Object, e As
                                        EventArgs)
```

per cui l'evento deve essere dichiarato come segue:

```
Dim WithEvents B As New Button()
Sub OnClick(sender As Object, e As EventArgs)
                                      Handles B.Click
  MessageBox.Show("Hello World")
End Sub
```

In Visual Basic 9, invece, il collegamento tra il delegato ed il metodo che lo implementa è più ampio in quanto è possibile utilizzare,in quest'ultimo, parametri con tipi non esattamente uguali purché possano rappresentare i tipi di parametri del delegato.
Così la seguente dichiarazione sarà sempre valida:

```
Sub OnClick(sender As Object, e As Object)
                                      Handles B.Click
  MessageBox.Show("Hello World")
End Sub
```

in quanto il tipo *Object* può comunque rappresentare il tipo *EventArgs*.

INTERFACCE DINAMICHE

Nei linguaggi tipizzati possiamo utilizzare soltanto proprietà dichiarate nel tipo.
Ad esempio, l'espressione:

```
Dim imp As Impiegato = GetImpiegato()
imp.Stipendio=1000
```

sarà valida solo se, a monte, nella classe Impiegato esiste una proprietà o un campo Stipendio :

```
Public Class Impiegato
    Private _Stipendio As Integer
    Public Property Stipendio() As Integer
        Get
            Return _Stipendio
        End Get
        Set(ByVal Value As Integer)
            _Stipendio = Value
        End Set
    End Property
...
End Class
```

A volte però potrebbe essere necessario accedere a un membro di un oggetto anche senza conoscerne in anticipo il tipo, Visual Basic lo consente attraverso il cosiddetto *Late Binding*, o associazione ritardata, ovvero con l'opzione *Option Strict Off* impostata.
E' facile intuire dunque che un espressione con una sintassi del tipo:

```
Dim imp As Object = GetImpiegato()
imp.Stipendio=1000
```

è valida per il compilatore anche se non sa ancora che l'oggetto che valorizza la variabile imp possiede o meno una proprietà *Stipendio*. Tale tecnica però non è esente da rischi in quanto se, a runtime, l'oggetto non possiede davvero una proprietà *Stipendio* verrà generato un errore non previsto. Prendiamo una funzione che restituisce un oggetto che ha una proprietà Stipendio:

```
Public Function GetDipendente (ByVal id  As Integer,
                    tipoDipendente As String) As Object
        If tipoDipendente="d" Then
                Dim dir As
                    Dirigente=Dirigenti.GetItem(id)
                Return dir
        ElseIf tipoDipendente="i" Then
                Dim imp As Impiegato =
                        Impiegati.GetItem(id)
                Return imp
        Else
                Return Nothing
        End If
End Function
```

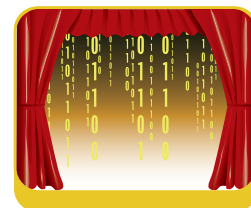
Con il Late Binding potremo utilizzare il risultato della funzione come:

```
Dim dipendente As Object = GetDipendente(1,"d")
Dim stipendio As Integer =
CInt(dipendente.Stipendio)
```

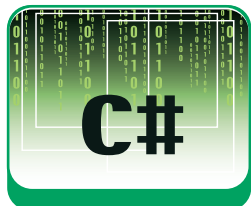
In Visual Basic 9, invece, è possibile definire un'interfaccia dinamica che rappresenti il tipo di dati restituit

```
Dynamic Interface IDipendente
  Property Stipendio As Integer
End Interface
Dim dipendente As IDipendente =
                                GetDipendente(1,"d")
Dim stipendio As Integer = dipendente.Stipendio
```

CONCLUSIONI

Come abbiamo visto, le novità sono davvero tante ed importanti.
Il momento in cui ci troveremo a programmare con il nuovo linguaggio, poi, non è tanto distante poiché esso sarà integrato nel nuovo Visual Studio che dovrebbe uscire nella prima metà del prossimo anno.

Maggiori approfondimenti sul nuovo VB possono essere trovati nel documento "Overview of Visual Basic 9.0" di Erik Meijer, Amanda Silver e Paul Vick (su cui è basato, in parte, anche questo articolo) pubblicato da Microsoft nel sito http://msdn.microsoft.com/vbasic/future/

PROGRAMMARE IN C++ SENZA "BACHI"

ALCUNI DEI BUG PIÙ INSIDIOSI CHE IL C++ PERMETTE DI INTRODURRE DERIVANO DA UNA GESTIONE INADEGUATA DELLA MEMORIA DINAMICA. IN QUESTA SERIE INTRODURREMO LE TECNICHE PIÙ AVANZATE PER SUPERARE IL PROBLEMA.

Quest'articolo è il primo di una piccola serie dedicata al Memory Management in C++. Si tratta di un argomento complesso e controverso, uno degli esempi più evidenti di quanto, nell'informatica, i problemi all'apparenza più insignificanti, se ignorati, possano trasformarsi facilmente in immani disastri, di quelli che fanno fallire un progetto intero. Ed è anche un'occasione per dimostrare ancora una volta che nel mondo della programmazione non esistono panacee, bensì una gran varietà di tecniche e soluzioni che il programmatore finale è tenuto a conoscere in prima persona, per poter scegliere fra di esse quella che più si adatta ai suoi bisogni. Alla luce di tutto questo, se programmate in C++ ma non avete mai sentito termini come smart pointers, RAII, exception safety, o pensate che: "i Garbage Collector sono roba Java", allora vi consiglio caldamente di prendervi del tempo per leggere questi articoli, e i riferimenti bibliografici ai quali essi rimandano. Approfondire questi argomenti oggi potrebbe, un giorno, semplificarvi molto la vita!

L'ORIGINE DI TUTTI I PROBLEMI

I problemi che analizzerò in questi articoli costituiscono una parte importante dei bug presenti nei programmi C++, e hanno tutti a che vedere con la gestione della memoria dinamica. Come sappiamo, infatti, in C++ è possibile istanziare gli oggetti ponendoli staticamente sullo stack, oppure dinamicamente sullo heap. Nel primo caso l'allocazione e la deallocazione sono gestite automaticamente dal compilatore, e seguono un ciclo di vita perfettamente predicibile, che va dalla dichiarazione della variabile fino a quando il flusso dell'esecuzione giunge fuori dalla sua area di visibilità (in gergo: out of scope), ed essa viene automaticamente distrutta. La creazione (e la distruzione) di una variabile in modo dinamico, invece, viene lasciata completamente sulle spalle del programmatore, che deve garantire per essa il rispetto di un ciclo di vita fondamentale (allocazione -> utilizzo -> deallocazione):

```
//Allocazione della risorsa
Oggetto* oggetto = new Oggetto();
//utilizzo della risorsa
oggetto->EseguiAzione();
//deallocazione della risorsa
delete oggetto;
```

I problemi legati alla memoria dinamica nascono tutti quando il programmatore fallisce nel rispettare questi tre passi: perché non ne compie qualcuno, o al contrario perché ne ripete qualcuno troppe volte, oppure ancora perché inverte l'ordine delle operazioni.

FALLE MNEMONICHE

Il primo problema che analizzeremo in questa sede si presenta quando il programmatore si dimentica di effettuare la deallocazione della risorsa. L'esempio più semplice, quindi, è:

```
Oggetto* oggetto = new Oggetto();
oggetto->EseguiAzione();
//...e la deallocazione?
return;
```

Questo caso dà origine a ciò che in gergo tecnico viene chiamato memory leak: l'applicazione riserva inutilmente dello spazio in memoria per una risorsa alla quale non farà mai più riferimento. Una piccola dose di risorse sprecate non costituisce un problema critico, ma quando ciò avviene in cicli, o in applicazioni destinate a girare a lungo, si ha dapprima in un rallentamento delle prestazioni, quindi l'esaurimento della RAM disponibile, e infine il crash dell'applicazione e/o del sistema. L'obiezione che: "nessun programmatore è così stupido da dimenticarsi una cosa del genere", o un'analoga variazione sul tema "basta stare attenti", non tiene conto della complessità

della programmazione reale, nella quale il flusso dell'esecuzione non è quasi mai lineare, ma è invece distribuito secondo la rete di relazioni che legano i vari oggetti – il che, spesso, rende impossibile stabilire a priori "quando", esattamente, una risorsa non sarà più utile e potrà quindi essere cancellata.

EXCEPTION SAFETY

Il flusso del programma, peraltro, può riservare delle belle sorprese anche quando appare perfettamente lineare. Prendiamo il codice seguente come esempio:

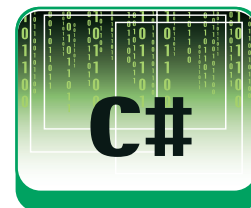
```
#include <iostream>
using namespace std;
class ErroreDiRiproduzione {};
class Musica {
public:
        string file;
        Musica(const string& ilFile) : file(ilFile) {
                std::cout << "risorsa allocata\n";
        }
        virtual ~Musica() {
                std::cout << "risorsa
                                deallocata\n";
        }
        virtual void Suona()
                   throw(ErroreDiRiproduzione) {
                if (!file.empty())
                        std::cout << "Sto
              suonando " << file.c_str() << endl;
                else

                  throw(ErroreDiRiproduzione());
        };
};
class Stereo {

public:
        void Esegui(const string& ilFile) {
                //Allocazione
                Musica* musica = new
                                Musica(ilFile);
                //Utilizzo
                musica->Suona();
                //Deallocazione
                delete musica;
        }
};
```

Nel metodo Stereo::Esegui, ho mantenuto scrupolosamente il parallelismo con il "ciclo di vita lineare" esposto qualche paragrafo fa. Possiamo quindi stare tranquilli? Non proprio, perché non abbiamo tenuto conto delle eccezioni. Proviamo a considerare questo caso:

```
int main() {
        try {
                Stereo stereo;
                stereo.Esegui("");
        }
        catch(...) {
        }


        return 0;
}
```

In questo caso abbiamo richiamato stereo.Esegui() passando una stringa vuota, cosa che genera un'eccezione. Al momento in cui l'eccezione viene sollevata, lo Stereo ha allocato un oggetto di tipo Musica, ma non l'ha ancora distrutto. Tuttavia l'esecuzione termina lì, lo stack viene srotolato, e il controllo viene restituito al gestore in main. Risultato: l'istruzione "delete musica" non sarà mai eseguita; pertanto, in un caso del genere otterremo un memory leak. Si dice, in gergo, che la funzione Stereo::Esegui non è exception safe. Potremmo pensare di rimediare con una gestione parziale dell'eccezione deallochi la risorsa:

```
void Stereo::Esegui(std::string file) {

Musica* musica = new Musica(file);
try {
musica->Suona();
}
catch {
        //dealloca in caso di errore
        delete musica;
        //rilancia l'eccezione
        throw;
}
//dealloca se tutto va bene
delete musica;
}
```

Così abbiamo ottenuto una funzione exception safe, ma a quale prezzo? Abbiamo complicato il codice, duplicando la deallocazione della risorsa e rendendo il sorgente brutto e poco leggibile. Peraltro, in C++, ogni funzione che non presenti una specifica delle eccezioni dichiaratamente nulla, ha il potenziale di generare eccezioni: questo insieme include la stragrande maggioranza delle funzioni tipiche, compresi operatori, assegnamenti e costruttori (anche quelli di oggetti temporanei, più difficili da controllare). Per seguire questo stile dovremmo stare costantemente all'erta, e imbottire il codice di blocchi try/catch. Una soluzione del genere, quindi, non è praticabile.

NOTA

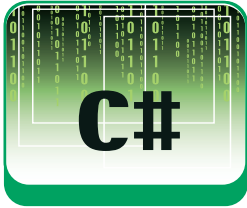
BOOST!

La libreria boost (www.boost.org) è un riferimento essenziale per i programmatori C++ più smaliziati, dal momento che fornisce molte funzionalità avanzate e colma alcune lacune del linguaggio. Molti smart pointer usati in questa serie vengono offerti da boost, e alcuni di essi sono stati inseriti nel TR1. È quindi probabile che diventino parte, in futuro, del C++ standard.

SUL WEB

Libreria Boost
www.boost.org

Libreria Loki
http://loki-lib.sourceforge.net/

Herb Sutter Guru of the Week
http://www.gotw.ca/gotw/

RAII

Una gestione meno ingenua del problema deve basarsi su un'evoluzione del semplice puntatore "nudo e crudo". Possiamo renderci conto facilmente dello stretto legame che unisce la vita dell'oggetto musica allo scope della funzione Stereo::Esegui. Ed è quindi immediato notare che, se musica fosse un oggetto, invece che un puntatore, sarebbe deallocato automaticamente in ogni caso, all'uscita dall'area di visibilità (riprenderemo questo punto più avanti).Può essere un'idea vincente, quindi, quella di incapsulare il puntatore in una classe e di associare la deallocazione della risorsa al suo distruttore. Questo tipo di tecnica ha un nome preciso: RAII (acronimo che sta per Resource Acquisition Is Initialization). Il principio fondante è che ogni acquisizione di risorse (che deriva da una chiamata all'operatore new) debba sempre coincidere con l'inizializzazione di un oggetto – cosicché la deallocazione della risorsa avvenga a discrezione del programma stesso, come semplice conseguenza dello stack unwinding.

NOTA

USARE BOOST

La libreria boost non è famosa per la semplicità d'installazione. Fortunatamente, le definizioni degli smart pointers non richiedono alcuna installazione: è sufficiente impostare l'IDE (o le variabili d'ambiente) inserendo fra le directory di inclusione la directory radice in cui si è scaricato boost.

BOOST:SCOPED_PTR

Quando la tecnica RAII si applica per la gestione generica di un puntatore, nasce un oggetto chiamato smart pointer (puntatore intelligente). Lo smart pointer più semplice che sia possibile realizzare è offerto dalla libreria boost (vedi riferimenti sui laterali), si chiama scoped_ptr, ed ha una definizione simile a quella che segue:

```
namespace boost {
  template<class T> class scoped_ptr : noncopyable {
    private:
          //puntatore "nudo e crudo"
          T* ptr;
    public:
          typedef T element_type;
          //costruttore (acquisisce la risorsa)
          explicit scoped_ptr(T* p = 0) : ptr(p) {};
          //distruttore (dealloca la risorsa)
          ~scoped_ptr() {delete p;}
          //operatori per la dereferenziazione
          T& operator*() const {return *ptr;};
          T* operator->() const {return ptr};
          T* get() const {return ptr};
          //operatore per la verifica di validità
          bool operator!() const {return ptr==0;}
          //Funzioni ausiliarie
          void reset(T* p = 0);
          void swap(scoped_ptr& b);
  };
}
```

Si tratta, come si può osservare facilmente, di un "vestito" molto semplice ed efficiente per un puntatore, grazie al quale possiamo riscrivere il nostro esempio in questo modo:

```
#include <boost/scoped_ptr.hpp>
using namespace boost;
//...definizioni varie...
void Stereo::Esegui(std::string ilFile) {
scoped_ptr<Musica> musica(new Musica);
musica->Suona();
}
```

La funzione è così ridotta a sole due righe, ed è perfettamente exception safe. Nella prima riga abbiamo inizializzato l'oggetto musica stavolta di tipo scoped_ptr<Musica>. Grazie all'overloading degli operatori, scoped_ptr imita il più possibile la sintassi di un puntatore. In questo modo, le seguenti istruzioni sono consentite indifferentemente sia se il tipo di musica è scoped_ptr<Musica>, sia se è Musica*:

```
musica->Suona();  //accesso ad un metodo o attributo
cout << *musica;  //dereferenziazione
if (!musica)                    //controllo di validità
        throw(ErroreDiRiproduzione());
```

L'accesso al puntatore originale, invece, deve essere richiesto attraverso l'invocazione del metodo get:

```
Musica* ptrMusica = musica.get();       //accesso
                                 diretto al puntatore
```

La funzione reset si occupa di acquisire una risorsa in un secondo tempo, distruggendo quella che deteneva in precedenza:

```
scoped_ptr<Musica> musica(new
Musica("Mozart.WAV"));                    //costruzione
musica.reset(new Musica("Beethoven.WAV"));
                                     //riassegnamento
musica.reset();       //riassegnamento a zero
```

Nell'esempio riportato qui sopra, il primo reset distrugge l'oggetto Musica("Mozart.WAV") costruito in precedenza, per allocare il nuovo Musica("Beethoven.WAV"). Questo stesso oggetto viene a sua volta deallocato nella seconda chiamata senza parametri, che rimette il puntatore "a terra" sul valore 0.
La funzione swap, infine, scambia i puntatori di due scoped_ptr:

```
scoped_ptr<Musica> musica1(new
                           Musica("Mozart.WAV"));
scoped_ptr<Musica> musica2(new
                        Musica("Beethoven.WAV"));
musica1.swap(musica2); //oppure swap(musica1,
                                      musica2)
musica1->Suona(); // "Sto suonando
```

- **MORE EFFECTIVE C++**
  Scott Meyers
- **EXCEPTIONAL C++ MORE EXCEPTIONAL C++**
  Herb Sutter
- **MODERN C++ DESIGN**
  Andrei Alexandrescu

```
                                        Beethoven.WAV"
musica2->Suona(); // "Sto suonando Mozart.WAV"
```

SEMANTICA DEL POSSESSO ESCLUSIVO

Per capire bene come funziona scoped_ptr è essenziale notare che questo eredita dalla classe boost::noncopyable, che impone costruttore per copia e operatore d'assegnamento privati, in questo modo:

```
class noncopyable
{
  protected:
        noncopyable() {}
        ~noncopyable() {}
  private:
        noncopyable( const noncopyable& );
        const noncopyable& operator=( const
                                   noncopyable& );
  };
```

In pratica, si tratta del modo più conciso ed esplicito per dire: "questa classe non può, e non deve, essere copiata". La ragione di questa scelta di design è semplice, ed è illustrata da questo caso ipotetico:

```
void Funzione {
        scoped_ptr<Musica> musica1(new
                  Musica("Mozart.WAV")), musica2;
        musica2 = musica1;
                                    //non si può fare!
}
```

Se l'istruzione "musica2 = musica1" fosse permessa, due diversi scoped_ptr si troverebbero a possedere puntatori uguali; all'uscita della funzione verrebbero richiamati entrambi i distruttori e la stessa risorsa verrebbe deallocata due volte di seguito. Quest'operazione prende il nome di double free, è uno dei peggiori pasticci che si possano combinare giocando con la memoria dinamica, e ha come conseguenza il crash dell'applicazione. Scoped_ptr risolve questi problemi impedendo ogni possibile operazione di copia: si dice in gergo tecnico che la semantica con cui opera è quella del possesso esclusivo della risorsa che detiene.

IMPIEGHI E LIMITAZIONI DI SCOPED_PTR

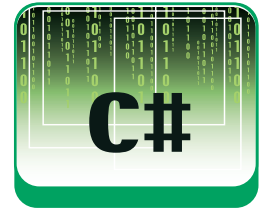
Giunti a questo punto, possiamo anche porci qualche domanda sul senso delle operazioni che stiamo studiando. Perché mai dovremmo aver bisogno di utilizzare uno smart pointer, puntandolo su una risorsa allocata dinamicamente, quando possiamo semplicemente usare un normalissimo oggetto?

```
void Stereo::Esegui(std::string ilFile) {
        Musica musica("");
        musica.Suona();
}
```

In effetti non c'è ragione di farlo, e ciò mostra che l'esempio che abbiamo usato, per quanto chiaro, è in realtà piuttosto fine a se stesso. Se vogliamo un esempio più concreto, dobbiamo chiederci quando ha senso fare ricorso alla memoria dinamica, e la risposta è prevalentemente una: "quando si deve far sopravvivere un oggetto al di fuori dello scope in cui si trova". Ma ciò è fuori della portata di scoped_ptr proprio per definizione: giacché questo tipo di smart pointer non permette la copia, non c'è modo di far sopravvivere la risorsa allocata da uno scoped_ptr al di fuori dell'area di visibilità in cui viene creata.Pertanto, se è vero che siamo di fronte allo smart pointer più semplice e performante (sostanzialmente non ci sono differenze prestazionali fra l'uso di uno scoped_ptr e quello di un "vero" puntatore), è anche vero che scoped_ptr trova nella programmazione reale degli impieghi assai limitati. L'uso di questo tipo di smart pointer si restringe tipicamente ad un paio di pattern formali, fra i quali spicca il modo molto elegante di definire l'idioma pimpl.

STD::AUTO_PTR

Per riuscire a far sopravvivere una risorsa al di fuori dello scope, occorre introdurre un nuovo smart pointer, chiamato auto_ptr. Si tratta di una classe molto interessante per diversi motivi. Innanzitutto è l'unico smart pointer previsto dal C++ standard, ragion per cui non avremo bisogno di librerie particolari per utilizzarlo, e sarà sicuramente la più "portabile" fra le soluzioni che analizzeremo in questa serie. Vale anche la pena di far notare che l'implementazione di una piccola classe come auto_ptr (un minuscolo ingranaggio incluso nell'header <memory>) ha messo seriamente sotto scacco il comitato di standardizzazione del C++, che ne ha stilato a breve distanza di tempo ben tre versioni diverse, dal momento che ogni soluzione trovata sembrava presentare qualche punto debole. La scelta finale è ricaduta su un modello simile:

```
namespace std {
  template <class T> class auto_ptr {
   public:
        //costruttori, distruttori, e assegnamento
        explicit auto_ptr (T* = 0);
        auto_ptr (const auto_ptr<T>&);€
```

PIMPL

Il pattern Pimpl, altrimenti noto come idioma handle-body è un sistema usato per nascondere l'implementazione di una classe separandola dal suo file header. Lo scopo è solitamente quello di velocizzare i tempi di compilazione.

Per ogni richiesta/ critica/suggerimento l'autore può (e deve!) essere contattato all'indirizzo articoli@robertoallegra.it

```
        void operator= (const auto_ptr<T>&);
        ~auto_ptr ();
        //operatori
        T& operator* () const;
        T* operator-> () const;
        T* get () const;
        //funzioni
        T* release();
        void reset (T* = 0);
  };
}
```

Grazie a scoped_ptr, che ci ha guidato fin qui, il nucleo di questa classe dovrebbe essere semplice da capire. La vera differenza sta nel fatto che auto_ptr prevede un costruttore per copia, un operatore d'assegnamento e la funzione membro release.

SEMANTICA DEL TRASFERIMENTO DEL POSSESSO

Il metodo release ordina allo smart pointer di restituire un puntatore alla risorsa che sta detenendo e resettare il proprio, mettendolo a terra. Si tratta, in sostanza, di uno "scaricabarile", grazie al quale auto_ptr si sbarazza della risorsa che detiene, e lascia che se ne occupi qualcun altro, che dovrà assumersi l'onere di eliminarla. Per questo motivo, è evidente che invocare la funzione release ignorandone il valore di ritorno è un errore, che genera un irreparabile memory leak:

```
#include <memory>
//... definizioni varie...
int main() {
        auto_ptr<Musica> musica(new
                        Musica("Mozart.WAV"));
        musica.release(); //rilascia la risorsa
        //ora musica punta a 0
        //e chi deallocherà più "Mozart.WAV"?
        return 0;
}
```

Accedere ad un auto_ptr subito dopo che si è invocata la funzione release, invece, è un errore ancora più grave, che manderà in crash l'applicazione:

```
auto_ptr<Musica> musica(new
                        Musica("Mozart.WAV"));
Musica* ptr = musica.release(); //ora musica
                                    punta a 0
musica->Suona()    //0->Suona? Crash!
delete ptr;
```

Infine, è possibile utilizzare il metodo release per travasare il puntatore di un auto_ptr in quello di un altro auto_ptr.

```
auto_ptr<Musica> musica1(new
                        Musica("Mozart.WAV")),
                    musica2;
//trasferisce la risorsa in musica2
musica2.reset(musica1.release());
```

L'operazione espressa qui sopra è esattamente la stessa che si ottiene effettuando una costruzione per copia, o un assegnamento (pertanto, in genere si preferisce questa sintassi, più semplice):

```
auto_ptr<Musica> musica1(new
                        Musica("Mozart.WAV")),
auto_ptr<Musica> musica2 = musica1;
//ora musica1 punta a 0
//mentre musica2 detiene la risorsa
```

Finalmente siamo arrivati al punto focale. Mentre scoped_ptr impediva ogni forma di copia, auto_ptr ne permette una, sebbene piuttosto peculiare. Per evitare il problema del double free che abbiamo visto precedentemente, infatti, la copia di un auto_ptr causa un'operazione di scaricabarile. Detto con parole più tecniche: la semantica con cui opera auto_ptr è quella del trasferimento del possesso della risorsa che detiene. In questo modo, si è sicuri che una risorsa sarà deallocata una (e una sola) volta.

MODELLO SOURCE/SINK

Proviamo ora a descrivere una variazione sul tema, che abbia effettiva necessità di memoria dinamica. Ipotizziamo di estendere il nostro lettore Stereo in modo che sia capace di operare una decodifica differente, a seconda dell'estensione del file. Dalla classe base Musica possono così nascere classi derivate d'ogni tipo, come ad esempio:

```
class MusicaMP3 : public Musica
{
public:
        MusicaMP3(const string& ilFile) :
                        Musica(ilFile) {};

        void Suona() throw(ErroreDiRiproduzione) {
                if (!file.empty())
                        std::cout <<
        "Decodifico e suono " << file.c_str() << endl;
                else

                        throw(ErroreDiRiproduzione());
        }
};
```

Ora scriviamo una funzione CreaMusica che, partendo dal nome del file, sappia costruire il giusto oggetto, e restituire un puntatore di tipo base Musica (nel

gergo dei pattern: una funzione Factory). In questo caso, non possiamo usare un semplice oggetto restituito per copia, perché taglierebbe inesorabilmente il comportamento polimorfico. E neanche possiamo passare un puntatore ad un oggetto temporaneo, dal momento che questo sarebbe deallocato alla fine della funzione, e passeremmo l'indirizzo ad un oggetto già distrutto! Abbiamo per forza bisogno di restituire un puntatore a memoria dinamica, o ancor meglio un auto_ptr.

```
auto_ptr<Musica> CreaMusica(const string& ilFile) {

        //ricava l'estensione del file
        string estensione(ilFile.substr(ilFile.length()
                                                   - 3));

        //crea la classe a partire dall'estensione
        if (estensione == "MP3")
                return auto_ptr<Musica>(new
                                      MusicaMP3(ilFile));
        else
                return auto_ptr<Musica>(new
                                          Musica(ilFile));
}
```

La funzione che abbiamo appena scritto può anche essere definita una source, categoria che include tutte quelle funzioni (o quei metodi) che restituiscono un auto_ptr. Lo smart pointer restituito ha vita molto breve, infatti viene usato per trasferire immediatamente il possesso all'auto_ptr (o allo scoped_ptr, che è sostanzialmente un const auto_ptr!) della funzione che lo richiama, detta sink – nel nostro caso Stereo::Esegui.

```
class Stereo {
public:
        void Esegui(const string& ilFile) {
                                          //Funzione sink
                auto_ptr<Musica>
                        musica(CreaMusica(ilFile));
                musica->Suona();

        }
};
```

Abbiamo così ottenuto un codice pulito e dal comportamento exception safe, dal momento che siamo sicuri che ogni risorsa allocata finirà sempre in un auto_ptr, e sarà quindi sempre deallocata.

```
int main() {
        Stereo stereo;

        stereo.Esegui("Mozart.MP3"); //Decodifico
                                     e suono Mozart.MP3
        stereo.Esegui("Bach.WAV");  //
                                     Sto suonando Bach.WAV

        return 0;
}
```

IMPIEGHI E LIMITAZIONI DI AUTO_PTR

Come abbiamo visto, auto_ptr si rivela di grande aiuto quando si ha a che fare con il passaggio di un oggetto creato da una classe source, in una sink. La ragione per cui preferire questo tipo di modello può essere un vincolo tecnico (come quello dell'esempio: preservare il polimorfismo), ma anche prestazionale. Poiché, infatti, passare una variabile temporanea per riferimento è un'operazione illegale, l'unica possibilità è la copia, che per oggetti di grosse dimensioni implica un notevole dispendio di tempo e risorse. Al contrario, il passaggio di un auto_ptr è un'operazione invariabilmente "leggera", dal momento che consiste solo nella copia di un puntatore. Anche questo smart pointer ha, tuttavia, i suoi limiti. La semantica del trasferimento del possesso rende auto_ptr una classe anomala, che non presenta alcune caratteristiche solitamente date per scontate, come l'equivalenza di una copia rispetto all'oggetto originale. Ciò implica che è sempre un errore grave utilizzare auto_ptr nei contenitori standard, dal momento che le copie temporanee che questi effettuano per le proprie operazioni portano tipicamente alla perdita (e al leak) della risorsa. Altro importante caveat: è un errore utilizzare auto_ptr e scoped_ptr per puntare ad array, dal momento che il distruttore di queste classi effettua sul puntatore un'operazione di delete, e non di delete[]. Per scoped_ptr la libreria boost fornisce la classe scoped_array, fatta apposta per allocare e deallocare vettori. Purtroppo, non esiste un equivalente auto_array per auto_ptr, quindi la soluzione più semplice è di evitare gli array primitivi, e puntare ad uno std::vector allocato dinamicamente (il che rappresenta comunque una miglior pratica di programmazione).

CONCLUSIONI

In questo primo articolo abbiamo fatto conoscenza con diversi elementi fondamentali che permettono una programmazione robusta ed exception safe. Rimangono ancora le limitazioni esposte nel paragrafo precedente: soprattutto l'impossibilità di usare i contenitori, e di far condividere a più smart pointer la stessa risorsa. Come da tradizione enigmistica: "la soluzione nella prossima puntata!"

Roberto Allegra

ESTENDERE MICROSOFT SQL

SCRIVERE FUNZIONI ESTESE PER SQL SERVER NON È SOLO UNA PREROGATIVA DEL C++. MOSTRIAMO UNA SOLUZIONE ALTERNATIVA PER REALIZZARE L'EQUIVALENTE DELLE EXTENDED STORED PROCEDURE NEL PIÙ AMICHEVOLE AMBIENTE VISUAL BASIC 6

I database relazionali sono croce e delizia dei programmatori moderni. Deliziano certamente tutti coloro che sono affascinanti dal mondo dei dati, dalla potenza del linguaggio SQL e, soprattutto, dalla estrema pragmaticità dello strumento. D'altro canto gli scettici trovano questo sistema di interrogazione ancora primitivo ed incapace di eseguire della vera logica applicativa che, dunque, deve continuare ad essere demandata al codice di programmazione.
Ma se solo si potesse effettuare chiamate a codice binario, ad esempio a proprie librerie direttamente dall'interno del database, magari nelle query stesse o nelle stored procedure, forse i due mondi - quello del database e quello del codice di programmazione tradizionale - potrebbeo finalmente convergere.

EXTENDED STORED PROCEDURE CON SQL SERVER 7 E 2000

I principali motori di database consentono la chiamata di funzioni presenti in librerie binarie scritte dal programmatore. Nel caso specifico ci occuperemo di SQL Server. Probabilmente pochi sapranno che Microsoft ha fornito un meccanismo per estendere le stored procedure attraverso la scrittura di nuove stored procedure non in linguaggio T-SQL, ma bensì in forma di dll binarie scritte in C++. Questa tecnologia è detta Extended Stored Procedure e, grazie al potente wizard fornito dapprima con Visual C++ 6.0 e in seguito con il Visual C++ di Visual Studio .NET 2002 e 2003, è possibile realizzare un esempio funzionante di extended stored procedure in pochi istanti (**Figura 1**).
Il wizard fa tutto il lavoro: prepara lo scheletro di una dll Win32, predispone la funzione di esportazione richiesta dalle extended sp e prepara già un esempio completo del metodo che implementerà la logica della stored procedure. Osserviamone l'esempio basato su quello generato dal wizard e presente nei sorgenti allegati nella cartella *extended_sp_cpp*:

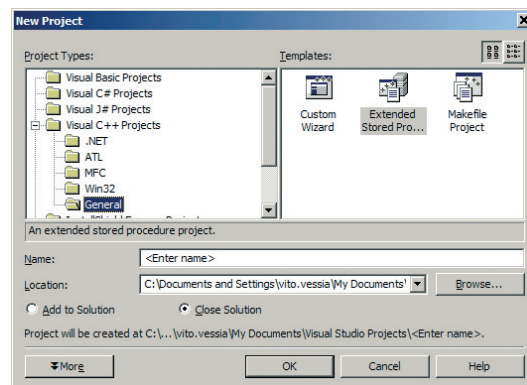



Fig. 1: *Wizard di Visual Studio .NET 2003 per la creazione di extended stored procedure in C++*

```
RETCODE __declspec(dllexport) xp_proc(SRV_PROC
                                        *srvproc)
{
   DBSMALLINT i = 0;
   DBCHAR colname[MAXCOLNAME];
         DBCHAR spName[MAXNAME];
         DBCHAR spText[MAXTEXT];
         // Name of this procedure
         _snprintf(spName, MAXNAME, "xp_proc");

         //Set up the column names
   _snprintf(colname, MAXCOLNAME, "ID");
   srv_describe(srvproc, 1, colname, SRV_NULLTERM,
     SRVINT2, sizeof(DBSMALLINT), SRVINT2, sizeof(DB
                                      SMALLINT), 0);
   _snprintf(colname, MAXCOLNAME, "spName");
   // Update field 2 "spName", same value for all rows
   srv_setcoldata(srvproc, 2, spName);
   srv_setcollen(srvproc, 2,
                  static_cast<int>(strlen(spName)));
         // Send multiple rows of data
   for (i = 0; i < 3; i++) {
                  // Update field 1 "ID"
                  srv_setcoldata(srvproc, 1, &i);
                  srv_setcoldata(srvproc, 3, sp
                                             Text);
     srv_setcollen(srvproc, 3, static_cast<int>
                                   (strlen(spText));
```

```
            // Send the entire row
    srv_sendrow(srvproc);
  }
        // Now return the number of rows processed
        srv_senddone(srvproc, SRV_DONE_MORE |
    SRV_DONE_COUNT, (DBUSMALLINT)0, (DBINT)i);
        return XP_NOERROR ;
}
```

Certo, con pochi clic un esempio funzionante è già pronto ed è sufficiente copiare la dll generata nella directory \Binn di SQL Server ed eseguire la registrazione della nuova stored procedure in SQL Server con il comando:

```
sp_addextendedproc 'nome_storedprocedure', 'nome
                                    libreria.DLL'
```

Si è così finalmente pronti ad utilizzare la nuova stored procedure, come mostrato in **Figura 2**.
D'altro canto, dalla produzione dell'esempio funzionante proposto in automatico dal template del wizard alla realizzazione di una propria stored pro-

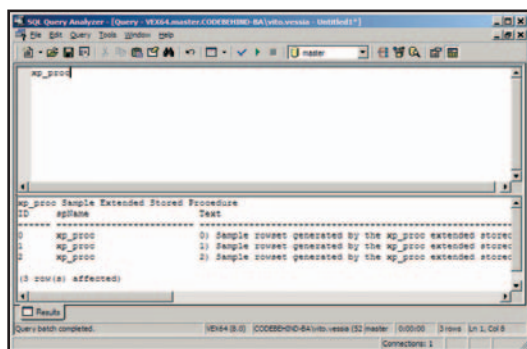



Fig. 2: ***L'extended stored procedure in azione***

cedure da zero in C++ il passo non è così breve. Se poi si considera che non tutti conoscono il C++ o vogliono avere a che fare con il livello di complessità di una soluzione del genere, diventa anche più interessante trovare una soluzione alternativa. Proviamoci…

CHIAMATE AD OGGETTI COM DA T-SQL

Il database Microsoft SQL Server 7.0 e 2000 offre la possibilità di effettuare chiamate COM ad oggetti che supportano l'interfaccia di automazione. A queste funzioni COM, via T-SQL, è possibile passare valori scalari (numeri, stringhe, date e quant'altro ma non interi record) ed è possibile ottenere come risultato un valore scalare.
È sostanzialmente quanto si può fare anche con le extended stored procedure, ma da queste è possibile ottenere al ritorno anche record come per le stored procedure tradizionali. Questa limitazione delle chiamate COM potrebbe sembrare rilevante, ma non è poi così grave. Infatti non è pensabile che in C++ o in qualsiasi altro linguaggio si costruiscano interi record, quando in T-SQL con semplici query è immensamente più facile fare altrettanto. È invece pensabile che ad una funzione esterna si deleghi il controllo di una password, un calcolo complesso o una qualsiasi altra logica che, infine, preveda la restituzione di un valore scalare piuttosto che di intere righe.
È certamente arrivato il momento di procedere con un esempio reale. In T-SQL è certamente possibile effettuare chiamate COM ad oggetti già presenti nel sistema, ma è altresì vero che si può pensare di scrivere propri oggetti COM che verranno richiamati da T-SQL. Ed è quello che faremo scrivendo una semplice DLL COM in Visual Basic 6 (**Figura 3**).

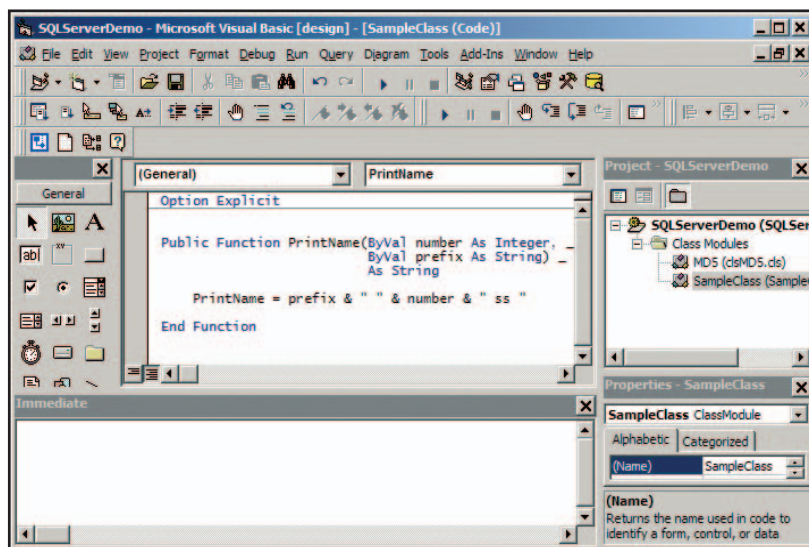



Fig. 3: ***La nostra classe Visual Basic 6***

Si tratta della "complessissima" classe *SampleClass* definita nella libreria *SQLServerDemo* e che pertanto è definita dal ProgId *SQLServerDemo.SampleClass*. Essa espone il metodo PrintName che accetta in ingresso due parametri number (di tipo intero) e *prefix* di tipo stringa e restituisce una stringa così composta: *prefix & " " & number & " ss "*:

```
Public Function PrintName(ByVal number As Integer, _
                    ByVal prefix As String) As String
    PrintName = prefix & " " & number & " ss "

End Function
```

Gli esempi di codice possono essere più o meno intelligenti e questo lo è di sicuro… ma sicuramente è sufficiente al nostro scopo: mostrare l'interazione da T-SQL. La prima operazione da effettuare è la creazione dell'istanza dell'oggetto COM. Eccone la sintassi:

```
sp_OACreate progid, | clsid,   token_oggetto OUTPUT
```

```
                    [ , contesto ( 1= inprocess, 2=locale(.exe) ]
```

Osserviamo un estratto di codice T-SQL:

```
declare @ServerID INT    --identificativo dell'istanza
declare @OLEResult int   --Hresult della chiamata
declare @errorSource INT  --numero di errore
declare @errorDescription varchar(220)  --descrizione
                                              dell'errore
EXEC @OLEResult = sp_OACreate
          'SQLServerDemo.SampleClass', @ServerID OUT
if @OLEResult <> 0
BEGIN
          EXEC sp_OAGetErrorInfo @ServerID,
       @errorSource OUTPUT, @errorDescription OUTPUT
END
```

In *@ServerID* finirà il riferimento all'istanza appena creata con la funzione di sistema sp_OACreate e verrà utilizzato per l'invocazione dei metodi dell'istanza. *@OLEResult*, invece, conterrà il valore *HResult* della chiamata, infatti, in caso di valore diverso da 0, e quindi di errore, sarà possibile chiamare la funzione *sp_OAGetErrorInfo* che restituisce proprio il numero e la descrizione precisa dell'errore:

```
sp_OAGetErrorInfo [ token_oggetto ]    [ , source
                      OUTPUT ]    [ , description OUTPUT ]
   [ , helpfile OUTPUT ]    [ , helpid OUTPUT ]
```

Queste informazioni finiranno nelle variabili *@erroreSource* ed *@errorDescription* Naturalmente questa eccezione può anche essere prodotta applicativamente all'interno del nostro codice Visual Basic 6. Se l'istanziazione è andata a buon fine, possiamo effettuare la chiamata al metodo *PrintName* della nostra dll con la seguente sintassi:

```
sp_OAMethod token_oggetto,     nome_metodo
                               [ , valore_ritorno OUTPUT ]
    [ , [ @nome_parametro = ] valore_parametro
                                  [ OUTPUT ]    [ ...n ] ]
```

Ed ecco come effettuare la chiamata al nostro metodo *PrintName*:

```
declare @prefixToPrint INT   --prefisso da passare alla
                                       funzione Visual Basic
declare @number int              --numero da passare
                                  alla funzione Visual Basic
declare @retString varchar(8000  --variabile che
                    conterrà il valore di ritorno della funzione
set @prefixToPrint = 'prova'
set @number = 15
--esecuzione della chiamata al metodo PrintName
                          della classe Visual Basic di esempio
EXEC @OLEResult = sp_OAMethod @ServerID, 'Print
         Name', @retString OUTPUT, @number = @numTo
                             Print, @prefix = @prefixToPrint
print @retString  --il risultato atteso sarà:   prova 15
                                                          ss
```

Abbiamo dichiarato le due variabili che conterranno i valori da passare al metodo PrintName, essere veranno passate con la sintassi *@nomeParametro = valore*, dove *@nomeParametro* è proprio il nome del parametro così come definito nella classe Visual Basic 6. Per il valore di ritorno, invece, è sufficiente aggiungere il postfisso *OUTPUT*. Si noti che non è necessario passare i parametri nell'ordine previsto dalla funzione da chiamare perché la convenzione di passaggio è basata sui nomi e non sulla posizione. È l'equivalente di usare la seguente sintassi cara ai programmatori Visual Basic:

```
Dim o As Object
Set o = CreateObject("SQLServerDemo.SampleClass")
Dim value as String
value = o.PrintName( prefix:= "ciro", number:= 544 )
```

Tornando alla nostra chiamata da T-SQL, se non si sono verificati errori, comunque tracciabili da una successiva chiamata a *sp_OAGetErrorInfo*, è possibile leggere o stampare il valore di ritorno della funzione VB6. Terminata la chiamata alla classe, non ci resta che rilasciare correttamente le risorse (l'istanza stessa della classe) al fine di evitare fastidiosi *memory leak*. Di questo si occupa un'altra stored procedure di sistema, la *sp_OADestroy*:

```
sp_OADestroy objecttoken
```

Ed eccone l'uso nel nostro esempio reale:

```
EXEC @OLEResult = sp_OADestroy @ServerID
```

COM IN BREVE

COM, acronimo per Component Object Model, è la tecnologia di definizione di oggetti e classi che consente l'interazione e la chiamate delle stesse a livello binario. Prima dell'avvento di .NET era sostanzialmente l'unico modo per condividere librerie di classi tra più programmi, magari scritti in linguaggi eterogenei purché COM compliant, senza disporre del sorgente. Questa tecnologia propone un modello di OOP non puro, ma basato unicamente sul polimorfismo di interfaccia. Tutte le interfacce COM ereditano dall'interfaccia di base IUnknown che offre i servizi di test delle interfacce supportate (il metodo QueryInterface), di allocazione e di rilascio dell'oggetto (AddRef e Release). Con l'introduzione di Visual Basic 4 e delle successive versioni, allo scopo di semplificare la fruizione e lo sviluppo COM da questo ambiente, è stata introdotta una nuova modalità di chiamata e di interazione con gli oggetti COM, dettà interfaccia di automazione. Gli oggetti che supportano questa variante devono implementare l'interfaccia IDispatch. Gli oggetti che supportono entrambe le modalità sono detti duali. Per un approfondimento sulla tecnologia si rimanda all'enorme materiale disponibile in rete.

Per effetto della *location transparency*, possiamo senza nessun problema effettura il debug della classe Visual Basic 6 direttamente dalla chiamata T-SQL. Sarà suffciente aprire il sorgente del progetto della DLL dall'IDE di Visual Basic 6 (**Figura 4**), mettere l'IDE in run con l'opzione *Wait for components to be created*, impostare un breakpoint all'inizio del metodo *PrintName*, effettuare la chiamata della funzione da T-SQL e attendere che Visual Basic 6 si attivi con l'istruzione col brealpoint i bella mostra...

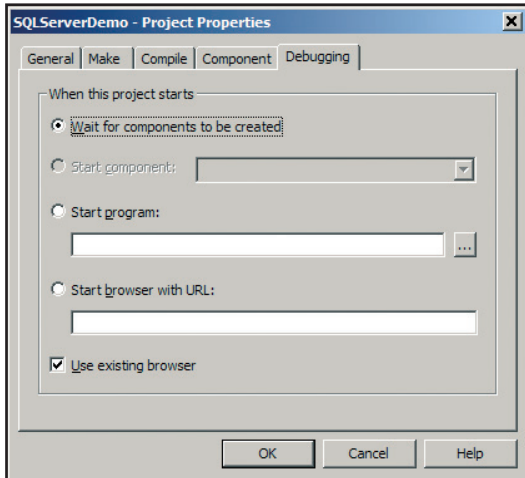



Fig. 4: ***Impostazione di debugging della DLL Visual Basic 6***

INCAPSULAMENTO DELLA CHIAMATA IN UNA USER FUNCTION T-SQL

Sebbene il meccanismo fin qui mostrato sia realmente molto potente e permetta di ottenere risultati straordinari con poco sforzo, potrebbe risultare un po' macchinoso data la complessità della sintassi prevista. Con l'approccio alla C++ la chiamata ad una extended stored procedure è indistinguibile dalla chiamata ad una stored procedure tradizionale. E allora perché non provare a semplificarsi la vita?
In SQL Server è prevista la possibilità di scrivere delle proprie funzioni richiamabili da query, stored procedure e, più in generale, da T-SQL. Ecco un esempio di user function che esegue la somma di due numeri:

```
CREATE FUNCTION dbo.MySum
(
        @num1 float,
        @num2 float
)
RETURNS float
AS
BEGIN
        return @num1 + @num2
END
```

Dovremo creare questa funzione all'interno di un database SQL Server, dopo di che ne sarà possibile la chiamata:

```
print dbo.MySum(2.3, 5.9)
--il risultato atteso è:  8.2
```

Potremo anche chiamare la funzione da una query:

```
select Campo1, dbo.MySum(Campo2, Campo3) as
                                        Somma from Tabella
```

E allora perché non creare una funzione che incapsuli la chiamata alla nostra PrintName? Detto fatto:

```
CREATE FUNCTION dbo.GetNumber
(
        @numToPrint int,
        @prefixToPrint varchar(255)
)
RETURNS varchar(255)
AS
BEGIN
        declare @OLEResult int,
        @ServerID INT,
        @retString varchar(8000),
        @errorSource INT,
        @errorDescription varchar(220)
        EXEC @OLEResult = sp_OACreate 'SQL
            ServerDemo.SampleClass', @ServerID OUT
        if @OLEResult<>0
        BEGIN
                EXEC sp_OAGetErrorInfo
    @ServerID, @errorSource OUTPUT, @errorDescription
                                                OUTPUT
                RETURN -1
        END
        EXEC @OLEResult = sp_OAMethod @
    ServerID, 'PrintName', @retString OUTPUT, @number
            = @numToPrint, @prefix = @prefixToPrint
        IF (@OLEResult <> 0)
        BEGIN
        EXEC sp_OAGetErrorInfo @ServerID,
    @errorSource OUTPUT, @errorDescription OUTPUT
                RETURN @errorDescription
        END
        ELSE
        BEGIN
                RETURN @retString
        END
        EXEC @OLEResult = sp_OADestroy
                                            @ServerID
        IF @OLEResult <> 0
        BEGIN
                EXEC sp_OAGetErrorInfo
    @ServerID, @errorSource OUTPUT, @errorDescription
                                                OUTPUT
```

```
                RETURN @errorDescription
        END
RETURN @retString
END
```

Questa funzione così come tutti gli altri esempi fin qui mostrati, è stata creata all'interno di una versione modificata del database Northwind di SQL Server. Un backup di questa versione accompagna i sorgenti allegati. Pertanto tutte le prove verranno effettuate puntando a questo database. Ed ecco, infatti, un'esempio di chiamata alla nostra funzione in una query:

```
select OrderID, CustomerID, dbo.GetNumber(OrderID,
                                    's') as CampoCOM from orders
```

Beh, è tutt'altra cosa rispetto alla complessità vista in precedenza. Naturalmente la chiamata a questa funzione potrà essere incapsulata in un'altra funzione:

```
CREATE FUNCTION dbo.GetNumberProxy
(
        @numToPrint int,
        @prefixToPrint varchar(255)
)
RETURNS varchar(255)
AS
BEGIN
        RETURN dbo.GetNumber(@numToPrint,
                                'Proxy ' + @prefixToPrint)
END
O in una stored procedure, colmando finalmente il gap
                                rispetto alle estendend sp:
CREATE PROCEDURE GetOrdersEx
        @MinimunFreight int = 0
AS
select orderid, shipname, freight, dbo.GetNumber
                Proxy(orderid, shipname) as CampoFunzione
from orders where freight >= @MinimunFreight
GO
```

IDENTIFICAZIONE E CHIAMATA DI OGGETTI COM

Ciascun elemento COM è identificato da un Guid (Global Unique Identifier) che è una sequenza di 16 byte assolutamente univoca perché generata a partire da informazioni peculiari dell'hardware della macchina da cui è stato prodotto. Per le classi COM esso assume il nome di CLSID (Class Id). Inoltre gli elementi COM sono definiti anche da un nome simbolico, detto ProgId (Programmatic Id) composto come nome_libreria.nome_interfaccia (es. ADODB.Recordset). Ogni chiamata ad oggetti COM restituisce sempre un numero intero che rappresenta lo stato del risultato che descrive la riuscita o meno dell'invocazione. Questo risultato è detto HResult (Handle Result) e assume il valore 0 in caso di esito positivo. Diversamente il numero resituito rappresenta proprio il tipo di eccezione occorsa.

UN ALTRO ESEMPIO: UN METODO ALTERNATIVO DI AUTENTICAZIONE DEGLI UTENTI

Ci sono tanti modi per realizzare l'autenticazione all'interno delle proprie applicazioni, dall'accesso ad active directory, all'uso degli utenti di SQL Server stesso, ma quello sicuramente più in voga consiste semplicemente nel predisporre una propria tabella di utenti nel database, con tanto di username e password (in chiaro o criptata) e, ogni qualvolta si debba autenticare un utente, si procede con una select in questa tabella per estrarre il record corrisponde allo username da validare, verificarne la password e restituire la risposta. Se la password è in chiaro, la verifica della password può essere fatta dal database stesso semplicemente aggiungendo la password come condizione di where:

```
select * from users where username = 'username' and
                                password = 'password'
```

Questo permette di ridurre gli accessi al database. Se la password è invece cripatata, magari con un solito algoritmo di hashing (ad esempio MD5 di RSA) che ha il vantaggio di essere irreversibile e di produrre chiavi hash sempre della stessa lunghezza indipendentemente dalla lunghezza della password, non è più possibile fare tutto con la query, ma si dovrà ottenere la chiave hash dal database e verificarne la bontà dal codice dell'applicazione. Ed ecco che la tecnica della chiamata a funzioni COM da T-SQL può venirci in aiuto consentendoci di ottenere l'approccio a singola select anche se le password sono criptate. Nel progetto Visual Basic 6 di esempio è riportata una seconda classe, chiamata MD5, e contenuta nel file clsMD5.cls. Essa espone il metodo pubblico:

```
Public Function DigestStrToHexStr(SourceString As
                                        String) As String
```

Questi accetta in ingresso una stringa e restituisce la chiave hash MD5 corrispondente. Il codice della classe è stato realizzato e messo a liberamente a disposizione in rete da Robert Hubley **[1]** e fa proprio al caso nostro. Abbiamo così realizzato la funzione T-SQL che ne incapsula la chiamata; di seguito se ne riporta l'header:

```
CREATE FUNCTION dbo.MD5Hash
(
        @stringToHash varchar(255)
)
```

```
RETURNS varchar(255)
```

Nel solito Northwind modificato, abbiamo aggiunta una tabella degli utenti:

```
CREATE TABLE [TSUSER] (
        [TSUSER] [int] NOT NULL ,
        [USERNAME] [nvarchar] (31) ,
        [PASSWORD] [varchar] (32) ,
        [FULLNAME] [varchar] (255) ,
        CONSTRAINT [PK_TSUSER] PRIMARY KEY
                                    CLUSTERED
        (
                [TSUSER]
        ) ON [PRIMARY]
) ON [PRIMARY]
GO
```

In essa conserveremo le utenze e le chiavi hash MD5 delle password. Ora non ci resta che definire una stored procedure che sia in grado di validare un utente a partire dallo username e che verifichi anche la password passata effettuando il match MD5 con la chiave hash contenuta nella tabella per quell'utente:

```
CREATE PROCEDURE Login
        @Username as varchar(50),
        @Password as varchar(50)
AS
        select * from tsuser
        where username =@Username and
            password = dbo.MD5Hash(@Password)
GO
```

Alla stored procedure dovranno essere passati lo username e la password. Se sono corretti, la sp restuirà proprio il record relativo alla login, diversamente non verranno restituite righe e pertanto l'utente si potrà considerare non autenticato. L'incapsulamento di tutta la logica di login in una sola stored procedere e comunque all'esterno del codice di programmazione, offre innumerevoli vantaggi in termini di versatilità e di separazione del codice ed è sicuramente uno degli approcci più puliti ed efficaci per risolvere il problema dell'autenticazione degli utenti.

CONCLUSIONI

Abbiamo visto come aggiungere un nuovo formidabile strumento alla già sovraffollata ed efficace cassetta degli attrezzi di SQL Server. La tecnica mostrata è sicuramente una valida alternativa a quella ufficiale basata sulla extended stored procedure e consente di estendere SQL Server anche con strumenti alla portata di tutti come Visual Basic 6. In alternativa si possono scrivere le librerie COM con Delphi, .NET o con lo stesso Visual C++, tutti ambienti che consentono la creazione di oggetti COM di automazione. Con SQL Server 2005 è prevista una nuova modalità di scrittura di extendend stored procedure direttamente da .NET; questa nuova tecnica consentirà la creazione di estensioni con la stessa semplicità vista nell'esempio di questo articolo, ma prevede due importanti limitazioni: la disponibilità di SQL Server 2005 e l'uso esclusivamente dell'ambiente .NET. Intanto, però, possiamo procedere sin da subito all'uso della nostra nuova tecnica casereccia…

RIFERIMENTI

[1] MD5 Message Digest Algorithm for Implementing Digitial Signatures – Robert Hubley - http://www.freevbcode.com/ShowCode.Asp?ID=741

Vito Vessia è cofondatore della codeBehind S.r.l. (http://www.codeBehind.it), una software factory di applicazioni enterprise, web e mobile, dove progetta e sviluppa applicazioni e framework in .NET, COM(+) e Delphi occupandosi degli aspetti architetturali. È autore del libro "Programmare il cellulare", Hoepli, 2002, sulla programmazione dei telefoni cellulari connessi al PC con protocollo standard AT+. Può essere contattato tramite e-mail all'indirizzo vvessia@katamail.com.

Vito Vessia

LOCATION TRANSPARENCY DI COM E DLL HELL

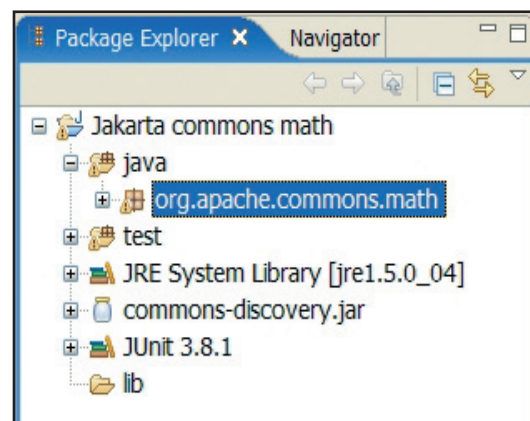
Una delle caratteristiche più entusiasmanti di COM è sicuramente la "location transparency", che significa letteralmente trasparenza dalla posizione (fisica del file). In pratica quando si effettua una chiamata COM non è necessario conoscere l'ubicazione fisica della dll, dell'ocx o dell'eseguibile che espone il servizio che si va a chiamare. Infatti le chiamate COM passano sempre attraverso il registry del sistema operativo che conosce l'ubicazione fisica del file. La posizione viene scritta nel registry durante l'operazione definita di registrazione della libreria, necessaria al momento dell'installazione nel sistema. Questa caratteristica è però un terribile effetto collaterale tanto noto agli utilizzatori di Windows: il fenomeno definito come DLL Hell, cioè inferno delle DLL. Se un programma distribuisce ed installa una componente COM, ad esempio una DLL, dopo la registrazione smetterà di preoccuparsi dell'ubicazione fisica della dll e semplice sfrutterà l'infrastruttura di COM per effettuarne la chiamata. Ma cosa accade se un secondo programma installa un'altra versione, magari precedente, della stessa dll? Anch'esso copierà da qualche parte una sua versione della stessa DLL e la registrerà nel sistema e ne effettuerà la chiamata senza alcun problema. Ma cosa accade alla prima applicazione? Per effetto della location transparency, quando effettuerà la chiamata alla sua cara DLL, in reatà Windows manderà in esecuzione la versione più vecchia installata successivamente dal secondo programma. Gli effetti sono devastanti e sotto gli occhi di tutti… In pratica, due oggetti COM con lo stesso CLSID (o anche con lo stesso ProgId) non possono coesistere contemporaneamente nel registry di Windows, anche se fisicamente risiedono in posizioni diverse, pertando la seconda registrazione sostuirà sempre la prima.

FATTI AIUTARE DA ECLIPSE

ECLIPSE CONTIENE CENTINAIA DI PICCOLE FUNZIONALITÀ NASCOSTE CHE RENDONO LA PROGRAMMAZIONE DI GROSSE APPLICAZIONI JAVA UN VERO PIACERE. ECCO COME DIVENTARE INCREDIBILMENTE EFFICIENTI IN MENO DI UN'ORA.

Per seguire questo articolo vi conviene giocare con un progetto reale. Per questo motivo abbiamo costruito un progetto Eclipse basato sul sorgente della libreria *Jakarta Commons Math*, una popolare libreria matematica open source per Java (*http://jakarta.apache.org/commons*).
Scompattate da qualche parte il file *progetto_eclipse.zip* che trovate sul CD e vedrete, nella risultante cartella, i file *.project e .classpath* che contengono la configurazione del progetto. Aprite Eclipse e scegliete *File->Import*. Apparirà un lungo elenco di cose che è possibile importare. Scegliete *General->Existing Project into Workspace* e premete *Next*. Con il tasto *Browse* selezionate la directory *Jakarta Commons Math* che avete ottenuto scompattando il file *.zip*. Attivate la checkbox *Copy projects into workspace*. Premete su Finish. Dopo qualche istante, vedrete il nuovo progetto nel workspace di Eclipse. Ora potete cancellare sia il file *.zip* che la cartella scompattata – Eclipse dovrebbe aver copiato tutti i file nel workspace.

ANATOMIA DI UN PROGETTO JAVA

Se esaminate il progetto nella vista *Package Explorer*, vedrete subito sei componenti: tre sottocartelle e tre librerie.
Cominciamo dalle librerie. Una è l'indispensabile runtime di Java. La versione di Java che vedrete dipende da quali JVM avete nel vostro sistema, e

quale tra queste viene usata da Eclipse come default. Potete anche aprire la libreria e scoprire da quali file è composta. Nel mio caso la *JRE System Library* contiene sei file .jar (che di fatto sono dei file *.zip* che contengono classi Java). Una seconda libreria è il popolare framework di test JUnit. JUnit è talmente diffuso che Eclipse lo propone come libreria standard alla pari delle librerie di Java. Il terzo file di libreria è *commons-discovery.jar*, la versione già compilata e impacchettata di un altro dei progetti Jakarta Commons. Dato che *Math* usa a sua volta *Discovery*, abbiamo aggiunto questo file alla directory *lib* e lo abbiamo registrato nel progetto come libreria. Trattandosi di una libreria, Eclipse la mostra nella root del progetto, e la cartella *lib* appare vuota.
Restano altre due cartelle, marchiate con un pacchettino marrone. Queste cartelle contengono i sorgenti del progetto. La cartella java contiene il codice di produzione, e la cartella test contiene i test basati su Junit. Eclipse ricompila automaticamente il contenuto di queste cartelle quando ce n'è bisogno, e mette il risultato (tipicamente un file *.class* per ciascun file *.java*) in una cartella di output. Nel caso di questo progetto, la cartella di output si chiama *bin*. Non potete vederla nella vista del Package Explorer, che si concentra sui sorgenti, non sui file compilati. Per vederla aprite la vista Navigator dal menu *Window->Show View*, o sem-

INSTALLARE ECLIPSE

Per installare Eclipse, accertatevi di avere sulla macchina il Java SDK (o almeno il Java Runtime Environment) e scompattate il file ZIP di Eclipse nella root del disco. Al primo avvio Eclipse chiederà un workspace, la directory che contiene le impostazioni e il codice dei progetti. Qualsiasi directory vuota va bene. Se non esiste, Eclipse provvederà a crearla.

plicemente usate l'explorer di Windows. Il Package Explorer mostra la struttura logica del progetto, il Navigator mostra la struttura fisica dei file che lo compongono. La configurazione del progetto è in una serie di file i cui nomi iniziano con un punto, come *.project* o *.classpath*. Per default, questi file non vengono mostrati nemmeno nel Navigator. Per vedergli dovete andare nell'explorer di Windows, o cambiare le impostazioni del Navigator.

Per cambiare le impostazioni di una vista, premete la piccola freccetta verso il basso nell'angolo in alto a destra della vista. Provateci con il Package Explorer. Due opzioni utili sono *Filters*, che decide cosa viene visualizzato e cosa no, e *Package Presentation*, che decide se la struttura dei package deve essere mostrata in modo "piatto" o come un albero.

Se volete una panoramica completa del progetto, o volete aggiungere, modificare o eliminare i suoi componenti, potete cliccare con il tasto destro sul progetto nel Package Explorer e scegliere *Properties*, l'ultima voce del menu contestuale. Da qui potete accedere a uno o due miliardi di possibili opzioni. Le prime opzioni che vi conviene guardare sono quelle della categoria *Java Build Path*. Da qui si possono gestire le cartelle dei sorgenti, quella dell'output, le librerie e le dipendenze tra i vari progetti nel workspace.

LE MANI NEL CODICE

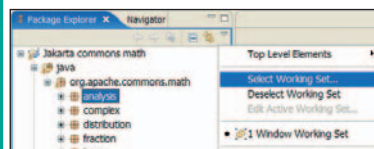
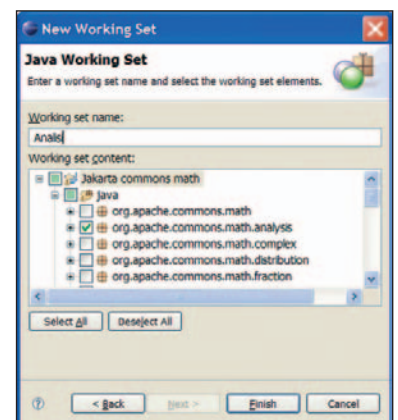
Fate doppio clic su un file sorgente Java per aprire l'editor. Oltre a tutte le funzionalità che ci aspettiamo, come la sintassi colorata e il completamento automatico, l'editor di Eclipse ha molte altre frecce al suo arco. Alcune tra queste non sono ovvie, ed è possibile che abbiate usato Eclipse per un po' senza scoprirle. Non sapete cosa vi state perdendo! Cominciamo dalle cose più ovvie. Se premete *Ctrl+Spazio* in qualsiasi momento mentre state scrivendo, attiverete il completamento automatico. Niente di nuovo, ma il completamento di Eclipse è uno dei più sofisticati in circolazione. Non solo riconosce i nomi di tutte le classi, i metodi, i campi e le variabili locali, ma analizza il codice circostante e presenta i suggerimenti in modo "intelligente": scoprirete che spesso quello che vi serve è proprio in cima alla lista delle proposte. Eclipse ha anche molte scorciatoie utili per scrivere automaticamente degli "stub" di codice da riempire. Ad esempio, basta scrivere main e premere *Ctrl+Spazio* per scrivere automaticamente un metodo *main()*. Usando lo stesso sistema dopo aver scritto *for* si ottengono suggerimenti per i tipi più comune di loop, scrivendo *sysout* si ottiene subito la noiosa riga di codice che stampa sullo schermo, e così via. I suggerimenti sono attivi ovunque, persino nei commenti Javadoc. Cercate *completion* e *templates* nella finestra delle proprietà di Eclipse per configurare ogni dettaglio del completamento automatico. E mentre state scrivendo la chiamata ad un metodo, provate ad usare la combinazione *Ctrl+Shift+Space* per avere qualche suggerimento sui parametri che dovete passare.

Una delle funzionalità più flessibili dell'editor è il Quick Fix. Il suo primo compito è individuare gli errori nel codice, e suggerire alcuni modi per correggerli. Aprite una qualsiasi classe, andate in qualsiasi metodo e scrivete questa riga di codice:

```
new GregorianCalendar();
```

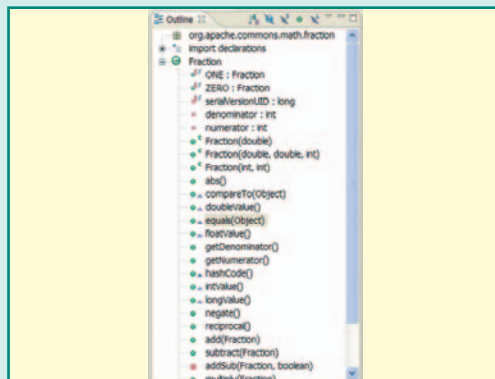
A meno che la classe che state usando per fare i vostri esperimenti non importi già *java.util.GregorianCalendar*, Eclipse segnalerà un errore. Cliccate sulla piccola icona rossa a fianco dell'errore (o premete il solito *Ctrl+1*) e avrete qualche suggerimento per rimediare automaticamente al problema. La soluzione più ovvia in questo caso è importare *java.util.Gregorian Calendar*, ma potete anche decidere di creare una nuova classe di nome *GregorianCalendar* nello stesso package. Correggere questo genere di problemi è così facili

MI INTERESSA SOLO QUESTO

In un progetto che contiene centinaia di classi, il Package Explorer può diventare affollato. Eclipse ha uno strumento fatto apposta per concentrarsi su "pezzi" di progetto: i working set. Poniamo ad esempio che ci interessi vedere solo il package org.apache.commons.math.analysis. Selezionate questo package nel Package Explorer, premete la freccetta in alto a destra della stessa vista e scegliete Select Working Set. Apparirà la finestra dei working set.

Premete su New per creare un nuovo working set. Scegliete Java e poi Next. Date un nome al working set, come ad esempio "Analisi", e controllate che sia selezionato solo il package analysis. Non siete costretti a selezionare un package: se volete potete anche scegliere una manciata di file sparsi in posti diversi del progetto.

Poi cliccate su Finish, selezionate il working set che avete appena creato e premete OK. Ora nel Package Explorer apparirà solo la roba che vogliamo vedere, e tutto il resto è stato nascosto. Eclipse ha anche aggiunto il nuovo working set al menu del Package Explorer, e selezionarlo è d'ora in poi una questione di secondi. Quando volete tornare a vedere tutto il progetto, selezionate il working set di default (Window Working Set).

NAVIGARE IN UN MARE DI CLASSI

Quando si lavora su un sistema object-oriented è importantissimo navigare le classi, i metodi e le loro relazioni. Niente di più facile.

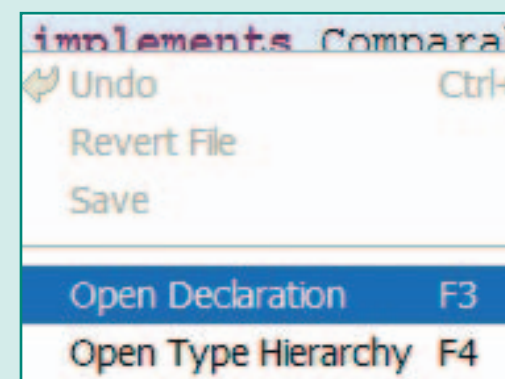
> UNA CLASSE DA VICINO

1 Cercate la classe Fraction (basta premere la combinazione di tasti Ctrl+Shift+T e scrivere le prime lettere del nome della classe) e apritela nell'editor. Guardate la vista Outline sulla sinistra per avere uno sguardo d'insieme su metodi e campi della classe. Il colore della piccola icona distingue il tipo di accesso. La lettera S identifica i membri statici, la C i costruttori, e le piccole freccette verso l'altro indicano che un metodo è l'override o l'implementazione del metodo di un supertipo.

> ECCO LA MIA FAMIGLIA

2 Quando ci si deve districare tra più classi, un buon modo per cominciare è aprire la vista Hierarchy. Selezionate la classe Fraction nell'editor o nel Package Explorer e scegliete la voce Open Type Hierarchy dal menu contestuale (o premete F4). La gerarchia mostra tutte le superclassi, le interfacce e le sottoclassi della classe selezionata. Anche in questo caso le semplici icone in cima alla vista ci permettono di decidere quale parte della gerarchia vogliamo vedere.

> HANNO CHIAMATO PER TE

3 La vista Call Hierarchy permette di vedere tutti i metodi che chiamano un determinato metodo. Selezionate il metodo getDenominator() della classe Fraction e scegliete Open Call Hierarchy dal menu contestuale. Ciascun metodo chiamante può essere a sua volta aperto.

> E QUESTO COS'È?

4 Andate nella dichiarazione di Fraction e selezionate il nome della superclasse Number (una classe che fa parte delle librerie standard di Java).
Scegliete Open Declaration dal menu contestuale (o premete F3) per aprire il sorgente di Number.

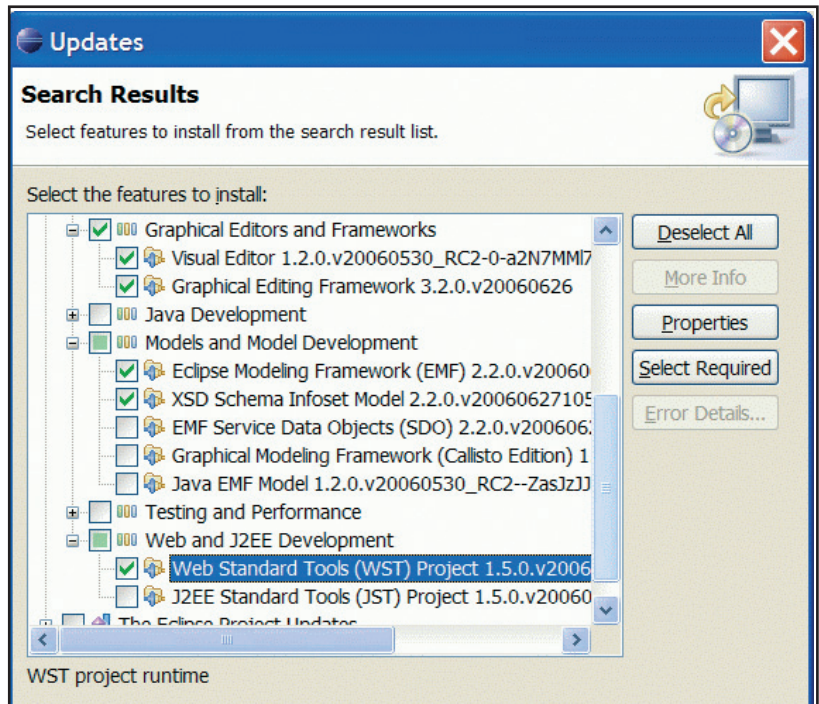
I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

che spesso conviene sbagliare volutamente: potete semplicemente scrivere i nomi dei metodi e delle classi che dovete ancora definire, chiedere ad Eclipse di crearli e scriverne l'implementazione. Un altro modo per andare veloci è ignorare cose come i downcast o i blocchi try-catch, e quando il compilatore protesta generarli al volo con il Quick Fix. Ora che abbiamo creato un nuovo *GregorianCalendar*, ci sono buone possibilità che desideriamo assegnarlo a qualche reference. Lasciando il cursore sulla stessa riga, premete *Ctrl+1*. Eclipse ci chiederà se vogliamo ad esempio assegnare l'oggetto ad una variabile locale, o ad un campo. Il sistema arriva addirittura a scegliere un nome piacevole per la nuova variabile. Quick Fix funziona in moltissimi altri casi: provate ad usarlo per compilare automaticamente i nomi dei parametri quando invocate un metodo, o addirittura (dopo aver abilitato l'opzione *Enable Spell Checking* nella sezione *Spelling* delle preferences) per avere suggerimenti ortografici quando scrivete i commenti e i nomi in inglese.

UN IDE SEMPRE PIÙ RICCO

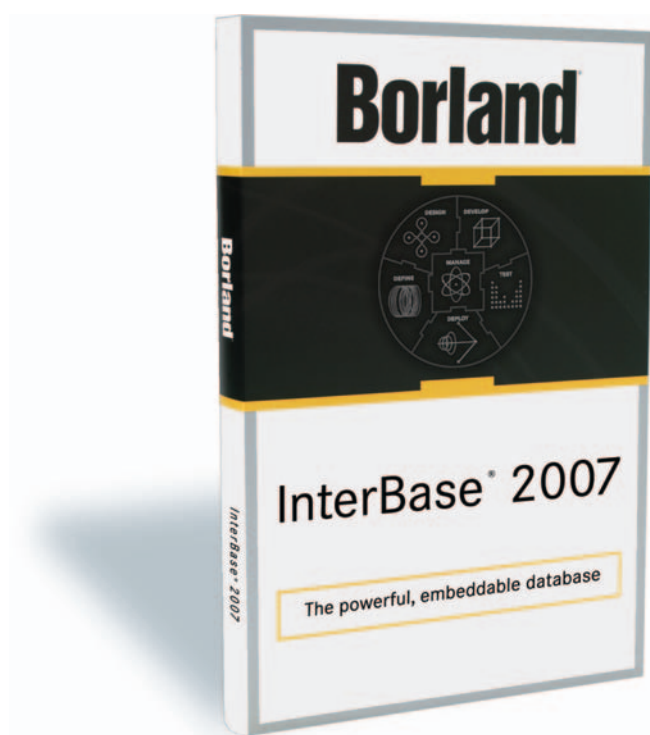
Ormai siamo abituati a pensare che ogni programma abbia le sue estensioni, che di solito si chiamano "plugin". Ci serve una funzionalità per disattivare i JavaScript in Firefox, o per ascoltare i file in formato Ogg Vorbis con Winamp? Basta installare un plugin, e il gioco è fatto. Be' Eclipse è diverso: non solo può essere esteso, ma è letteralmente *fatto* di estensioni. L'intero IDE è una collezione di plugin, tutti caricati da un minuscolo motore. Ad esempio, tutto il sistema di sviluppo Java è composto in realtà da una quarantina di plugin. Se ne avete voglia, potreste rimuovere tutte le funzionalità legate a Java e sostituirle con altre che permettono di sviluppare in qualche altro linguaggio. Il modo migliore di installare nuovi plugin è usare il meccanismo degli Update Site. Selezionate il menu *Help -> Software Updates -> Find and Install*. Apparirà una finestra che vi permette di scegliere se volete scaricare gli aggiornamenti per i plugin già installati, o installarne di nuovi. Selezionate *Search for new features to install* e premete su *Next*. Nella pagina successiva vedrete uno o più "siti di aggiornamento". Se vedete un sito che si chiama *Callisto Discovery Site*, scegliete quello (Callisto è una release contemporanea di Eclipse e di tutte le sue estensioni principali, e garantisce che non avrete problemi di compatibilità tra i vari plugin). In caso contrario, scegliete *The Eclipse Project Updates*. Poi cliccate su *Finish* e aspettate che Eclipse abbia curiosato nel sito di update che avete scelto. Se il sistema vi chiede di scegliere un "mirror", prendete pure quello che vi piace di più. Ben presto spunterà un alberello pieno di roba da installare. Con lo stesso meccanismo si possono aggiornare i plugin già esistenti, oppure andarne a cercare di nuovi al di fuori del sito di Eclipse. La maggior parte degli autori di plugin pubblicano il proprio sito di update, che potete facilmente aggiungere alla lista di Eclipse. Il vostro viaggio nel mondo di questo immenso IDE è appena cominciato!

Paolo Perrotta

NON PERDERE MAI NIENTE

Non c'è niente di peggio che modificare una serie di righe di codice in rapida successione per poi rendersi conto che hai appena cancellato per sbaglio ore di lavoro. Per lenire quei dolorosi momenti Eclipse include un semplice sistema di versionamento locale che arriva dove il semplice "undo" non può arrivare. Scegliete dal Package Explorer uno dei file che avete modificato di recente, e selezionate dal menu contestuale la voce Replace With->Local History. Apparirà un elenco di tutte le ultime modifiche al file, ciascuna con la sua data. Selezionatene una, e tornerete subito indietro nel tempo. Se volete vedere esattamente cosa avete modificato, selezionate Compare With->Local History per vedere subito quali sono le differenze tra il file attuale e tutte le sue versioni precedenti.

La History ci permette di programmare con animo sereno, sapendo che i nostri errori non saranno puniti severamente. Ma questa non è una scusa per non fare frequenti backup, o meglio ancora usare un sistema di versionamento!

INTERBASE SERVER 2007 IL NUOVO CHE C'ERA GIÀ

BORLAND E' TORNATA E LO HA FATTO ALLA GRANDE. OLTRE ALLE NUOVE VERSIONI DEGLI IDE E DEI COMPILATORI C#, DELPHI, C++, JAVA ECCO ARRIVARE UN DATABASE DALLE CARATTERISTICHE STRAORDINARIE. AFFIDABILITÀ, VELOCITÀ, MA ANCHE MOLTO ALTRO...

Prima di tutto un po' di storia. Interbase nasce intorno al 1996 come giusto corredo di quel Delphi che rappresentava la punta di diamante della programmazione agli inizi dell'era RAD. Già a quei tempi Interbase era un server avanzato, multiutente, con supporto alle transazioni. MySQL era appena un database dotato del minimo indispensabile per sopravvivere. Gli altri concorrenti proponevano soluzioni multiutente e con supporto alle transazioni solo in casi veramente eccezionali. E poi cosa è successo? Nel 1999 per motivi squisitamente commerciali Borland ha deciso di rendere OpenSource la propria versione di Interbase, dalla quale è nata il conosciutissimo PostgreSQL ad esempio. Interbase da quel momento in poi ha vissuto di fasi alterne, poiché Borland come tutti sappiamo aveva rivolto le proprie attenzioni al Life Cycle Management piuttosto che alla produttività individuale. Solo oggi, con il rilancio dei prodotti Turbo, la rinascita di una divisione apposita dedicata allo sviluppo e la prospettiva di tornare ad essere leader nel mondo degli IDE e dei compilatori, Borland ha tirato nuovamente fuori dal cilindro Interbase nella sua nuova fiammante versione "2007". Si tratta di un prodotto che, nonostante i suoi circa 10 anni di "purgatorio", arriva sul mercato denso di innovazioni che, come sempre accade quando Borland scende in campo, non mancheranno di essere imitate.

INTERBASE E I CONCORRENTI

Quando si analizzano pregi e difetti di un prodotto innovativo è utile confrontarlo con le caratteristiche di prodotti ben conosciuti. Nel nostro caso prenderemo in esame MySQL e Microsoft SQL Server. Ma semplicemente perché vogliamo mettere in luce i punti di maggiore interesse di questo database.

Prima di tutto il formato delle tabelle. MySQL prevede diversi "engine" per la gestione dei dati, in particolare InnoDB, MyISAM, BerkleyDB. Tuttavia MyISAM non gode del supporto alle transazioni, InnoDB è transazionale ma non supporta la ricerca Full Text, BerkleyDB supporta il lock dei dati solo al livello di tabella. Viceversa Interbase utilizza un unico Engine che espone tutte le funzionalità necessarie. Non ci sono scelte architetturali da compiere nell'avviare un progetto.

Stored Procedure e Triggers sono conquiste che MySQL ha compiuto solo con le versioni 5.x. Interbase ne fa uso estensivo fin dalle prime versioni.

Backup e disaster recovery. Tutti sanno che è abbastanza facile, nella maggior parte dei casi effettuare un backup di un DB MySQL. Questa è una funzionalità standard che in situazioni normali non comporta particolari difficoltà. Se però il vostro database è di dimensioni generose e/o è soggetto ad un numero elevato di INSERT/UPDATE ottenere delle copie di backup consistenti senza "Lockare" completamente o in parte il server per lungo tempo, diventa difficoltoso. Per quanto ri-

guarda Interbase esiste una funzionalità di "Online Backup" che consente di ottenere una copia "consistente" del vostro database aggiornata all'istante in cui la copia viene lanciata. In Interbase 2007 questa caratteristica è stata ulteriormente migliorata. E' stato aggiunto un sistema di journaling dei dati basato su Log che rende particolarmente efficiente il Backup dei dati e il loro recupero in caso di un problema Hardware o di un qualunque altro problema che generi una situazione di possibile perdita di informazioni. Anche la velocità di recupero è piuttosto elevata. Mentre nel caso di MySQL occorre un certo periodo di tempo per recuperare interamente un database, nel caso di Interbase il recupero è quasi immediato. Interbase utilizza un meccanismo che consente di agire su un singolo Bit per verificare il Roll Back di una transazione, rendendo il tutto particolarmente efficace.

Tipi di dati: una particolarità interessante è quella del concetto di dominio. Il dominio dei dati si può equiparare al concetto di classe derivata nella programmazione ad oggetti. Supponete che la vostra azienda voglia adottare un nuovo formato per l'identificativo delle merci. Il nuovo formato potrebbe essere più lungo o più corto di quello che state adottando al momento. La soluzione tipica sarebbe quella di andare in ciascuna tabella che fa uso del dato in questione e modificare il campo con una lunghezza omogenea a quella richiesta dalla nuova situazione. Viceversa in Interbase è possibile utilizzare i "DOMINI". Si definisce un dominio, ovvero un campo che gode di particolari proprietà, si creano i campi in tutte le tabelle "derivandoli" da quel dominio. Se si volesse cambiare la lunghezza di un dato, sarebbe sufficiente modificare il "Dominio Padre" per modificare a catena tutti i campi ad esso correlati.

Utenti e ruoli: In MySQL il controllo sull'accesso al database o alle tabelle, pur essendo granulare è delegato all'utente. Se avete un database contenente 200 utenti dovete settare i permessi per ogni singolo utente. In Interbase è previsto il concetto di "Ruolo" è cioè possibile dividere gli utenti in gruppi. Modificando il permesso di un gruppo lo si modifica contemporaneamente per tutti gli utenti che appartengono a quel gruppo.

Monitoring: Un ultimo accenno va fatto riguardo alla possibilità di "monitorare" il comportamento del server, riguardo al carico di lavoro e a tutti gli altri parametri che possono essere utili ad un sistemista per l'ottimizzazione del server. Interbase include un completo tool per il monitoring delle prestazioni, che consente rilevamenti molto più avanzati rispetto a quelli offerti da MySQL. E' ad esempio possibile visualizzare graficamente parametri come il numero di transazioni, il numero totale di Insert, di UPDATE e molto altro ancora.

INTERBASE E LA MULTIPIATTAFORMA

Rispetto a Microsoft SQL server, il primo parametro di differenza che si può notare è quello relativo alla disponibilità per i diversi sistemi. Interbase è disponibile per Windows, Linux e Solaris. Questo lo rende particolarmente appetibile anche in ambienti di Hosting. Esistono diversi connector poi che lo rendono utilizzabile con i diversi linguaggi. Per quanto riguarda il Web, sia PHP che .NET sono ampiamente compatibili con Interbase. Per le applicazioni standalone, gli strumenti di sviluppo che più si accordano con Interbase 2007 sono rappresentati dalle varie applicazioni della suite Turbo di Borland: Delphi, C#, C++. In ogni caso è anche possibile utilizzare il Microsoft Visual Studio in modo del tutto trasparente.

Per quanto riguarda una comparazione con Microsoft SQL Server, oltre alle caratteristiche di Cross Platform di cui abbiamo già parlato, sarebbe possibile scendere ulteriormente nel dettaglio parlando della velocità nel recupero di dati in modalità di failover, della gestione dei trigger e di altri dettagli squisitamente tecnici. Si tratta tuttavia di due piattaforme perfettamente paragonabili. Viceversa il paragone non è possibile se si parla di costi ed occupazione di risorse. Il costo di Borland Interbase 2007 è di 200$ per la licenza Server (1 CPU) e 150$ per accesso utente. Il costo della versione Desktop è di 60$ a postazione. Le versioni di MS Sql Server con le stesse caratteristiche hanno un costo decisamente superiore.

CONCLUSIONI

Il rientro di Borland nel settore dello sviluppo copre, con Interbase 2007, l'intero specchio degli strumenti indispensabili per un programmatore. Lo stile è quello di sempre. Alta produttività per quanto riguarda gli IDE, grandi performance e leggerezza per quanto riguarda i database. Nonostante che gli strumenti Turbo vengano indicati come adatti per la produttività individuale, in realtà se coniugati ad un server come Borland Interbase 2007 assumono il ruolo di entry point anche per lo sviluppo aziendale. Quando le esigenze cominceranno a crescere sarà sempre possibile passare alle versioni superiori. Interbase 2007 rimane tuttavia estremamente scalabile ed adatto fin da adesso alla produttività aziendale, individuale e persino ad essere messo in produzione su sistemi di hosting.

Sarà disponibile a breve dal sito www.borland.it una versione trial per 4 CPU e 20 accessi utenti. Per ulteriori informazioni scrivere a: italy.info@borland.com

SOFTWARE SUL CD

Turbo Delphi for .NET

UN RITORNO STORICO

Dieci anni fa Delphi ha segnato il punto di svolta nelle tecnologie di sviluppo. Nessuna innovazione ha segnato tanto quanto l'introduzione degli IDE RAD una rivoluzione nel campo dello sviluppo. E fu Borland con il suo Delphi a portare questa innovazione. Dopo anni in cui Borland si è quasi totalmente dedicata al segmento "business" ecco che torna con una suite di ambienti dedicati alla produttività individuale fra cui spiccano "Turbo C#" di cui ci siamo occupati nel numero scorso e "Turbo Delphi" di cui ci occupiamo in questo numero. Si tratta di prodotti eccezionali che meritano di essere installati sulle macchine di tutti gli sviluppatori. Nella versione turbo sono completamente completi e gratuiti e godono di caratteristiche di eccellenza che meritano di essere provate. Necessita registrazione sul web per ottenere la chiave di sblocco che lo abilita all'uso gratuito e completo per un tempo illimitato.

Directory:/turbodelphi

ANIMADEAD 2.0

L'ANIMATORE DI SCHELETRI

Diciamo che volete realizzare un'animazione 3D completa. Da dove partire? AnimaDead utilizza un approccio originale. Quello che dovete fare è disegnare uno scheletro del personaggio da animare, fatto di barrette e giunture. A questo punto programmate il movimento dei singoli pezzi tenendo conto di come sono congiunti fra loro. Infine ricoprite il tutto con i tessuti. L'idea è ben realizzata e l'approccio è veramente molto interessante. Su ioProgrammo ce ne siamo occupati già in passato con un articolo approfondito, la tecnica rimane decisamente affascinante.

Directory:/ Animadead

IRRLICHT 1.1

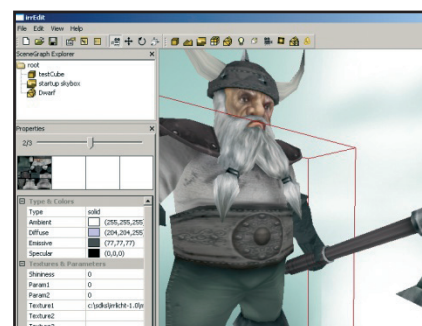
IL MOTORE 3D PER ECCELLENZA

In questo numero di ioProgrammo presentiamo XNA, il nuovo game engine di Microsoft, ma quali sono i suoi rivali? Sicuramente irrlicht è uno di questi! E' vero per usare irrlicht bisogna avere qualche nozione di C++ ma l'engine è talmente ben congegnato che la difficoltà tecnica è veramente bassa. Su ioProgrammo ci siamo occupati a lungo di Irllicht in passato e vi abbiamo persino dedicato un libro "Programmare VideoGiochi" di Alfredo Marroccelli

Directory:/ Irrlicht

$IRREDIT 0.5

IL DISEGNATORE DI SCENE

Ok programmare videogiochi con

Irrlicht è abbastanza semplice. Ma come si disegnano le scene? Come si creano i sofisticati ambienti 3D che rendono così ricchi i giochi moderni? Irredit à appunto un "disegnatore di scene 3D". Un supporto che i programmatori di Irrlicht hanno voluto regalare ai loro seguaci. Ottima idea! Se ne sentiva la mancanza.

Directory:/ Irredit

CRYSTAL SPACE 0.99

IL RE C'È ANCORA!

È stato per lungo tempo il dominatore

della scena degli Engine 3D. Lo è stato almeno fino all'avvento di irrlicht e del più recente XNA. Questo non vuol dire che non rappresenti ancora lo stato dell'arte nel campo della programmazione dei Videogiochi. Ad oggi si tratta di uno degli engine 3D che conta il maggior numero di installazioni e prodotti sviluppati. Sicuramente un prodotto da provare

Directory:/ crystalspace

APOCALYX 0.8.10

L'ENGINE 3D ALTERNATIVO

Basato su OpenGL si tratta di un engine completo e molto affidabile per la realizzazione di ambienti 3D. Gli sviluppatori sono molto attivi ed il progetto viene costantemente aggiornato, attualmente conta una base di utenti piuttosto ampia e le caratteristiche tecniche sono di tutto rispetto. Da provare!

Directory:/ Apocalyx

RUBY 1.8.5

IL VERO NUOVO CHE AVANZA

In America è il linguaggio che maggiormente ha scalato i vertici delle classifiche di utilizzo nell'ultimo anno. In Italia sta giungendo rapidamente come un ciclone a fare capolino nel panorama dello sviluppo. Bello, elegante, con una curva di apprendimento praticamente nulla, si tratta di un linguaggio di scripting che ottimamente si presta ad essere utilizzato sia sul Web sia in modo standalone. Recentemente ce ne siamo occupati in un bell'articolo di Paolo Perrotta, che mostrava come utilizzarlo per programmare un plugin per Skype messenger

Directory:/ Ruby

PYTHON 2.5

L'EX GIOVANE RAMPANTE

Python è stato considerato per lungo tempo il nuovo che avanza. Attualmente non lo si può più definire in questo modo, Python è ormai un linguaggio stabile e completo che trova applicazione in un gran numero di progetti. Se ne parla sempre di più in campo industriale come su Internet. Soprattutto un gran numero di applicazioni anche in ambiente Windows girano ormai grazie a Python e presentano interfacce grafiche ottimamente strutturate. Ciò nonostante Python rimane un grande linguaggio di scripting adatto a gestire in modo completamente automatico buona parte di un sistema operativo sia esso Linux o Windows

Directory:/ Python

ECHO 1.4

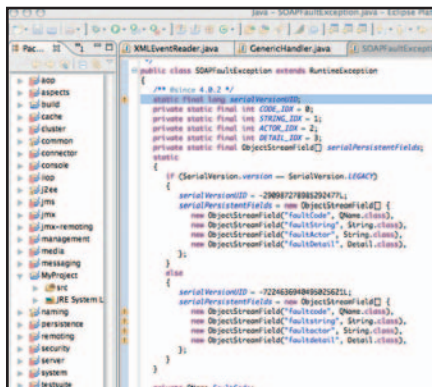
PROGRAMMAZIONE AJAX SECONDO JAVA

Ce ne parla in modo approfondito Roberto Sidoti in questo stesso numero di ioProgrammo. Echo è un framework sofisticato quanto semplice nell'utilizzo e che garantisce la creazione di applicazioni Web 2.0 compatibili in tecnologia Ajax, utilizzando Java. Non è cosa da poco se si pensa che ormai la direzione dello sviluppo sul Web è quella della migrazione verso Ajax. Certamente Java non può esimersi dal percorrere questa strada se non vuole rimanere arretrato proprio in un settore strategico come Internet

Directory:/ Echo

ECLIPSE 3.2.1

L'IDE TUTTOFARE!

Indispensabile per i programmatori Java, grazie ai suoi Plugin sta diventando rapidamente un punto di riferimento

importante per tutti i programmatori in quasi ogni linguaggio. Eclipse è un IDE nativo per Java ma che può facilmente essere esteso per diventare un ottimo ambiente anche per tutti gli altri compilatori. In questo numero ce ne parla in modo esauriente Paolo Perrotta nel bell'articolo che ne descrive alcune delle caratteristiche essenziali

Directory:/ Eclipse

JDK 1.5.0_09

AD UN PASSO DA JAVA 6

L'ennesima nuova release del compilatore unico per Java. Ormai Java 6 è alle porte e questa ulteriore release del JDK rappresenta semplicemente una piccola evoluzione del ben noto compilatore. Viceversa c'è molta attesa per Java 6 che dovrebbe rappresentare un vero punto di svolta per i programmatori di Sun

Directory:/ Jdk

PHP 5.1.6

IL LINGUAGGIO DEL WEB

Insieme ad Apache e a Mysql fa girare la stragrande maggioranza delle applicazioni internet moderne, soprattutto OpenSource. Si tratta di un linguaggio snello, elegante, completo, completamente ad oggetti ma che può essere utilizzato anche in modo procedurale. Recentemente ha acquisito nuovo spessore con il rilascio dello Zend Framework, che rende ancora più immediata la programmazione con PHP “costringendo” lo sviluppatore ad aderire al pattern MVC che con la sua logica di separazione dei vari layer di programmazione rappresenta sicuramente il pattern più utile ed utilizzato al mondo

Directory:/ PHP

MYSQL 5.0.26

IIL PRINCIPE DEI DATABASE

Indispensabile per programmare web application in tecnologia PHP. Non che non sia possibile utilizzare altri database, ma MySQL e PHP rappresentano veramente un binomio inscindibile. L'integrazione fra questo database e il linguaggio di scripting più usato sulla rete è talmente alta da fare divenire quasi un obbligo l'uso congiunto di questi due strumenti

^Directory:/ MySQL

PROGRAMMAZIONE GENETICA

NELL'AMBITO DEL CALCOLO EVOLUTIVO, LA PROGRAMMAZIONE GENETICA BASANDOSI SULLE LEGGI DI EVOLUZIONE NATURALE DELLA SPECE TENTA L'ARDUO COMPITO DI PRODURRE IN MODO AUTOMATICO PROGRAMMI PER LA RISOLUZIONE DI PROBLEMI

Il progetto è ambizioso: si tratta di produrre i programmi in modo automatico. Da programmatori dovremmo rifiutarci di studiare tali metodi, poiché lo scopo ultimo è quello di produrre dei programmi autoapprendenti e ciò equivarrebbe a eliminarci. Ma suppongo che questo non avverrà, almeno non a breve. Per contro è molto interessante tutta la teoria che affronta tale tematica. In questo appuntamento cercheremo di circoscrivere la vasta teoria che c'è dietro queste tecniche ad un ben definito esempio che abbiamo introdotto nello scorso numero e che riprenderemo da zero adesso.

L'ALFABETO DELLA GENETICA

La programmazione genetica è un estensione degli algoritmi genetici. Per questi ultimi si osserva una popolazione di individui o membri che evolve seguendo le basilari regole della selezione naturale come proposte da Darwin. In particolare, un individuo è codificato con una stringa binaria e l'evoluzione avviene sostanzialmente attraverso i due processi di mutazione (riproduzione asessuale) e ricombinazione o crossover (riproduzione sessuale). Da cui deriva la selezione e l'eventuale eliminazione. Gioca un ruolo fondamentale il fit del membro rispetto alla popolazione ossia la sua "adeguatezza", la cui misurazione è uno dei nodi cruciali di tali studi. Per cui si assegna un punteggio ad ogni membro, tale valore è conosciuto come il fitness e sarà alla base della selezione naturale. Da notare che non sempre è valida la famosa legge del più forte per la quale si predilige un grado di fitness elevato. Potrebbe essere un obiettivo dell'evoluzione la "varietà" con la quale si privilegerebbero popolazioni il più possibile eterogenee da un punto di vista genetico. La programmazione genetica funziona allo stesso modo ma focalizza l'interesse su popolazioni che siano programmi per computer. Così fissando un obiettivo, ossia un problema da risolvere si applicano le operazioni evolutive al fine di produrre il "miglior" programma che risolva quel problema. Il programma potrà variare dinamicamente ed essere rappresentato da schemi di parsing tipicamente ad albero. La selezione degli elementi su cui applicare la selezione influenza la convergenza dell'algoritmo. I principali metodi di selezione si basano: sul valore del fitness normalizzato; in funzione del fitness relativo e secondo un'estrazione casuale tra cui si sceglie il fitness migliore. I primi sono criteri più selettivi, e producono un più rapida convergenza verso la soluzione, mentre i secondi favoriscono la diversità e quindi offrono più possibilità di raggiungere la soluzione ottima.

LA PROGRAMMAZIONE AUTOMATICA

Come sottolinea Koza, il padre della programmazione genetica, una delle maggiori sfide dell'uomo è risolvere problemi senza l'apporto umano. In particolare potrebbe essere desiderabile produrre programmi indipendentemente dagli input e soprattutto dal problema stesso da risolvere. Koza ha stilato un decalogo, anzi meglio un esadecalogo, sedici regole che dovrebbe rispettare un sistema per la creazione automatica di programmi per computer. Esaminiamolo:

1. Partire da istruzioni di un linguaggio ad alto livello, specificando le richieste del problema;
2. Produrre un risultato nella forma di una sequenza di passi che possono essere eseguiti su un computer.
3. Produrre un'entità che può essere eseguita su un computer;
4. Avere l'abilità di determinare automaticamente l'esatto numero di passi che devono essere elaborati;
5. Avere la capacità di riutilizzare il codice prodotto qualora fosse necessario;
6. Avere l'abilità di riusare gruppi di passi con differenti istanze di valori (uso di parametri);
7. Avere l'abilità di usare le memorie interne nella forma di: singole variabili, vettori, matrici, stack, code, liste, e altre strutture dati;
8. Avere l'abilità di implementare: selezioni, cicli e ri-

corsioni;
9. Avere l'abilità di organizzare automaticamente gruppi di passi all'interno di gerarchie;
10. Capire se è necessario impiegare gli elementi al punto 7 e 8.
11. Avere l'abilità di costruire programmi simili a quelli che produce l'uomo includendo macro, librerie, tipi, puntatori, operazioni condizionali, funzioni logiche, funzioni intere, funzioni reali, input multipli, output multipli e istruzioni in codice macchina;
12. Operare in modo ben definito e distinguendo tra ciò che l'utente deve fornire e ciò che il sistema rilascia;
13. Essere indipendente dal problema, nel senso che non deve essere modificato per ogni nuovo problema;
14. Produrre soluzioni per un'ampia casistica di problemi;
15. Essere scalabile, ossia deve ben funzionare anche su grandi versioni del problema stesso;
16. Produrre risultati competitivi rispetto ai programmatori umani.

Si nota come i primi sono i requisiti di un normale programma, o meglio linguaggio di programmazione. Mentre gli ultimi, in particolare l'ultimo definiscono la vera peculiarità del sistema che stiamo studiando.

IL NOSTRO PROGETTO

Si tratta di un programma orientato agli oggetti sviluppato in C++ che tenta attraverso la produzione e la manipolazione di alberi di generare programmi che risolvano delle espressioni algebriche. Gli operatori di base introdotti sono unari e binari, e per essi sono state previste le basilari operazioni algebriche. È stata introdotta una classe base virtuale che permetterà di trattare i quattro tipi di nodi. Il codice C++ è:

```
// classe base per quattro tipi di nodi
class Node {
  protected:
    Node *parent;          // nodo padre
    int n_children;        // numero di nodi figlio
    int n_valid;           // flag di validità
  public:
    Node(void);            // costruttore
    ~Node(void);           // distruttore: resetterà il
                           // puntatore di
                           // questo nodo del padre
    int type;              // indica il tipo di nodo
    void invalidate_count(void); // chiamato quando il
                                 // numero di figli di questo
                                 // nodo è cambiato
    virtual Val value(void) = 0; // Per assegnare
                                 // valori a  questo nodo
    virtual int count(void) = 0; // restituisce il numero
                                 // di figli
    virtual Node *find(int) = 0;  // usata per restituire
                                  // un particolare nodo
    virtual char *print(void) = 0; // visualizza
                   // l'espressione sottoalbero come una stringa
    // resettta il nodo padre
    void set_parent(Node *n) {parent = n;};
    // accede al nodo padre
    Node *get_parent(void) {return parent;};
};
```

*parent è il puntatore al nodo genitore del nodo puntatore corrente (this). I due membri protetti *n_children* e *n_valid* indicono rispettivamente il numero di figli (ossia un sottoalbero) e un flag che innesca, quando necessario, una nuova computazione. Entrambi non possono essere qui usati, ma solo per classi derivate. La funzione *invalidate_count* viene richiamata quando cambiano il numero di figli, solitamente in seguito ad operazioni di crossover o riproduzione, tale informazione è resa disponibile da *n_valid*. Con *value()* si valuta un nodo a partire dai suoi figli. Per valutare l'intero albero è necessario sottoporre la funzione alla radice. Il membro *find()* trova un figlio nell'albero, mentre *print()* visualizza ricorsivamente il sottoalbero specificato. Ogni tipo di nodo è una sottoclasse della classe *Node*. Le due classi nodo non terminali sono: *Binary node* e *Unary_node*. Esse sono sostanzialmente uguali, variano di fatto solo per il numero di figli. Le funzioni aggiuntive presenti in questa classe sono introdotte principalmente per accedere ai campi privati e per l'autoesplorazione. I costruttori per ogni sottoclasse prendono come parametro un puntatore al nodo padre. L'operatore o il terminale sarà associato al nuovo nodo. Ecco il prototipo delle due classi:

```
// Il nodo delle operazioni binarie
class Binary_Node : public Node {
  private:
    Node *left;         // figlio sinistro
    Node *right;        // figlio destro
    Binary_Func *f;     // La funzione di questo nodo
  public:
    // Il costruttore prende il padre della funzione
    Binary_Node(Node *,Binary_Func *);
    ~Binary_Node(void);       // il distruttore
                              cancellerà il figlio
    // Aggiunta di un nodo a sinistra
    void set_left(Node *l) {left = l;};
    //  Aggiunta di un nodo a destra
    void set_right(Node *r) {right = r;};
    // Accesso al nodo figlio sinistro
    Node *get_left(void) {return left;};
    // Accesso al nodo figlio destro
    Node *get_right(void) {return right;};
    // Accesso alla funzione
    Binary_Func *get_func(void) {return f;};
    double fitness;        // fitness dell'albero (se è radice)
```

```
    // La funzione value ha necessità di passare il valore
    // al figlio di f
    Val value(void);         // funzione valore
    int count(void);         // funzione contatore
    Node *find(int);         // funzione trova
    char *print(void);       // funzione stampa
};
// Il nodo delle operazioni unarie
class Unary_Node : public Node {
  private:
    Node *child;             // il singolo figlio
    Unary_Func *f;           // la funzione di questo nodo
  public:
    // Il costruttore prende il padre della funzione
    Unary_Node(Node *,Unary_Func *);
    ~Unary_Node(void);       // Il distruttore
                                        cancellerà il figlio
    // Aggiunta di un nodo figlio
    void set_child(Node *c) {child = c;};
    // Accesso al nodo figlio
    Node *get_child(void) {return child;};
    // Accesso alla funzione
    Unary_Func *get_func(void) {return f;};
    // funzione value passa il valore del figlio di f
    Val value(void);         // funzione valore
    int count(void);         // funzione contatore
    Node *find(int);         // funzione trova
    char *print(void);       // funzione stampa
};
```

Il valore di questo tipo di nodi si ottiene semplicemente chiamando l'operatore con il valore di ogni figlio. Per esempio, il valore la funzione value della classe *Binary_node* è:

```
// funzione value – semplicemente valuta la funzione
                                        usando i valori
// figli sinistro e destro
Val Binary_Node::value(void)
{
    // Se il destro o il sinistro non esistono,
    // genera un errore e restituisce zero
    if ((!left) || (!right)) {
        fprintf(stderr,"A binary node is missing a child!\n");
        return (Val)0;
    }
    // return it
    return (f->eval(left->value(),right->value()));
}
```

I nodi terminali possono contenere sia costanti che variabili; queste sono rappresentate attraverso le classi *Terminal_const* e *Terminal_Var*. Le due classi sono leggermente differenti. Esaminiamo il codice commentato per approfondirne il loro funzionamento.

```
// Il nodo terminale di valore costante
class Terminal_Const : public Node {
    private:
    Val constant;            // Il valore costante
  public:
    // Il costruttore prende il padre e il valore
    Terminal_Const(Node *, Val);
    // La funzione restituisce una costante
    Val value(void) {return constant;};
    // La funzione per questo nodo restituisce 1
    int count(void) {return 1;};
    Node *find(int);         // funzione trova
    char *print(void);       // funzione stampa
};
// Il nodo terminale variabile.  Usa un reference a val per il
                                                    valore
class Terminal_Var : public Node {
  private:
    Variable *var;      // reference della variabile value
  public:
    // Il costruttore prende il padre e la variabile puntatore
    Terminal_Var(Node *, Variable *);
    // La funzione restituisce il valore della variabile
                                                  corrente
    Val value(void) {return *var->var;};
    // questa restituisce il puntatore al valore (per la copia)
    Variable *val_p(void) {return var;};
    // Per questo nodo la funzione è 1
    int count(void) {return 1;};
    Node *find(int);         // funzione trova
    char *print(void);       // funzione stampa
};
```

Se le classi e le funzioni finora esaminate sono di base e servono alla manipolazione e la ricerca nell'albero adesso vedremo come si può creare una popolazione, ossia un gruppo di programmi. La classe *Population* contiene: le funzionalità necessarie per creare un gruppo iniziale di programmi, per assegnare ad ognuno di essi un valore di fitness; per selezionare i membri per la riproduzione e per creare una nuova generazione usando regole di crossover. In un ulteriore modulo applicativo, saranno trattate le informazioni circa: il set degli operatori e delle variabili, le funzioni di fitness, e gli indici di riproduzione. Il prototipo della classe è il seguente

```
class Population {
  private:
    Binary_Func **b_funcs;      // Lista delle funzioni
                               //binarie disponibili
    int num_b_funcs;           // Numero di funzioni
                              // binarie disponibili
    Unary_Func **u_funcs;       // Lista delle funzioni
                                // unarie
    int num_u_funcs;           // Numero delle funzioni
                                // unarie
    Variable **vars;           // Lista delle variabili
    int num_vars;              // Numero di variabili
    Val (*const_rand)(void);    // Funzione per generare
```

```
                              // costanti random
    Binary_Node **trees;      // Lista di alberi nella
                              // popolazione
    int size;                 // Misura della popolazione
    // Tre funzioni che attraverso la ricorsione generano
    // sottoalberi casualmente sia con nodi operatore
                                        // unari che binari
    Node *new_node(Node *);
    void build_tree(Binary_Node *);
    void build_tree(Unary_Node *);
    // Usata per creare una nuova generazione
    void crossover(Binary_Node *,Binary_Node
                         *,Binary_Node **,Binary_Node **);
  public:
    // Il costruttore prende inizialmente la misura (size),
    // numero e lista delle funzioni binarie
    // numero e lista delle funzioni unarie, numero e lista
    // variabili e delle funzioni per trovare costanti random
   Population(int,Binary_Func **,int,Unary_Func
                                  **,int,Variable **,int,
             Val (*)(void));
    ~Population(void);        // Il distruttore pulisce la
                                                  memoria
    void sort(void);          // Ordinamento della
                                  popolazione per fitness
    void spawn(float);        // Creazione di una nuova
                                               generazione
    // configurazione del fitness per ogni membro
    void evaluate(double (*)(Node *));
    // restituisce la corrente misura della popolazione
    int members(void) {return size;};
    // restituisce l'espressione di un membro
    char *print(int);
    // restituisce il fitness di un membro
    double fitness(int i) {return trees[i]->fitness;};
};
```

Di seguito a titolo di esempio vedremo come sono implementati alcuni di questi membri e comunque troverete tutto il codice all'interno del CD allegato alla rivista. Approfondiamo l'aspetto della creazione di una nuova popolazione e la creazione di una nuova generazione. Il costruttore prende dei parametri come input, ossia: il numero di alberi della popolazione, la lista degli operatori e delle variabili e la funzione per la produzione casuale di costanti e crea una lista di nodi binari. Appunto del tipo *Binary_node* che rappresentano il nodo radice di ogni albero della popolazione. Un generatore di alberi casuali viene invocato per ognuno dei nodi appena descritti al fine di creare la popolazione iniziale. Per la creazione della nuova generazione si devono affrontare due stadi. Nel primo la popolazione corrente deve essere valutata per determinare il fitness di ogni membro; cosi, secondariamente, ai membri ritenuti adeguati è permesso riprodursi. È il modulo applicativo (ultimo paragrafo) che si occupa di fornire la funzione del fitness. Questa funzione è applicata alla popolazione usando *evaluate()*. La funzione *spawn()* produce una nuova generazione per la popolazione. Essa lavora seguendo i seguenti passi:

1. Ordina la popolazione per il valore di fitness (assegnato durante la fase di valutazione).
2. Sceglie gli alberi padre in base al momento della creazione;
3. Cancella gli alberi che saranno sostituiti dai figli. Scalando in modo ordinato dal basso verso l'alto della popolazione ordinata;
4. Crea il figlio usando la funzione di crossover;

La funzione di crossover elabora il compito della riproduzione dell'albero. Essa si basa su due padri. Per ognuno di essi si seleziona un nodo e si scambiano i sottoalberi identificati come radici dai nodi selezionati. I passi per realizzare ciò sono:

1. Copiare i due nodi padri ai figli (per conservare gli alberi padre);
2. Selezionare due nodi casualmente, evitando di scegliere la radice;
3. Eliminare il sottoalbero del figlio 1, e sostituirlo con una copia della sezione di crossover del figlio 2;
4. Ripetere per il caso opposto.

Tale metodo è stato esaminato nello scorso numero.

LE IMPLEMENTAZIONI

Lo spazio a disposizione non ci consente di esaminare tutte le implementazioni delle funzioni incontrate. Ne osserveremo qualcuna particolarmente significativa, per le altre si rimanda alla codice riportato sul CD. Cominciamo con un esempio di operatore aritmetico.

```
// funzione per sommare due valori
Val add(Val a, Val b)
{
   return (a+b);
}
// il carattere + indica la somma
char add_sign[] = " + ";
// la rappresentazione in funzione binaria
Binary_Func Add = {add, add_sign};
```

Proseguiamo con una funzione della classe *Node*, in particolare esaminiamo la funzione che conta i nodi binari.

```
// contatore di nodi binari
int Binary_Node::count(void)
{
   // se il contatore corrente è valido restituisce
                                              il valore
   if (n_valid)
```

```
    return n_children;
  // altrimenti ricalcola
  // somma sopra il sinistro e il destro (se esistono)
  n_children = 0;
  if (left)
    n_children += left->count();
  if (right)
    n_children += right->count();
  // conta questo nodo
  n_children++;
  // adesso questo è valido
  n_valid = 1;
  return n_children;
}
```

Dal modulo popolazione vediamo come si implementa l'importante processo di crossover, così come descritto.

```
void Population::crossover(Binary_Node *p1,
                           Binary_Node *p2, Binary_Node **c1,
              Binary_Node **c2)
{
  // copia il primo padre
  *c1 = (Binary_Node *)tree_copy(p1,NULL);
  // copia il secondo padre
  *c2 = (Binary_Node *)tree_copy(p2,NULL);
  // casualmente seleziona il numero di nodo
                              dal padre 1(punto di crossover)
  int cp1 = int_rand((p1->count()-1)) + 2;
  // ottieni il nodo (selezione di crossover)
  Node *cs1 = p1->find(cp1);
  // casualmente seleziona il numero di nodo
                              dal padre 2(punto di crossover)
  int cp2 = int_rand((p2->count()-1)) + 2;
  // ottieni il nodo
  Node *cs2 = p2->find(cp2);
  // cancella la selezione dal primo figlio e copia sul
                                              secondo padre
  //
  // trova il nodo che potrebbe essere cancellato dal
                                                primo figlio
 Node *delnode = (*c1)->find(cp1);
  // conserva questo padre
  Node *parent = delnode->get_parent();
  // se il padre è un nodo binario, risulta fuori se
                                         questo era un figlio
  // sinistro
  int left=0;
  if (parent->type == NODE_BINARY)
    if (((Binary_Node *)parent)->get_left() ==
                                                     delnode)
      left = 1;
  // cancella il sottoalbero
  delete delnode;
  // fa una copia della seconda selezione di crossover
                                                per il padre 2
  Node *newnode1 = tree_copy(cs2,parent);
  // resetta i puntatori al padre
  // nel caso in cui sia unario:
  if (parent->type == NODE_UNARY)
    ((Unary_Node *)parent)->set_child(newnode1);
  // nel caso in cui sia binario
  else
    if (left)
      ((Binary_Node *)parent)->set_left(newnode1);
    else
      ((Binary_Node *)parent)->set_right(newnode1);
  //
  // fase di canncellazione dal secondo figlio e link al
                                               // primo padre.
  // trova il nodo che potrebbe essere cancellato dal
                                                // primo figlio
  delnode = (*c2)->find(cp2);
  // conserva questo padre
  parent = delnode->get_parent();
  // se il padre è un nodo binario, risulta fuori
                                       // se questo era un figlio
  // sinistro
  left=0;
  if (parent->type == NODE_BINARY)
    if (((Binary_Node *)parent)->get_left() ==  delnode)
      left = 1;
  // cancella il sottoalbero
  delete delnode;
  // fa una copia della selezione di crossover del padre 2
  Node *newnode2 = tree_copy(cs1,parent);
  // resetta il puntatore padre
  // nel caso in cui l'operatore è unario
  if (parent->type == NODE_UNARY)
    ((Unary_Node *)parent)->set_child(newnode2);
  // nel caso in cui l'operatore è binario
  else
    if (left)
      ((Binary_Node *)parent)->set_left(newnode2);
    else
      ((Binary_Node *)parent)->set_right(newnode2);
  return;
}
```

ESEMPIO DI APPLICAZIONE

Per terminare esaminiamo una possibile applicazione. Consideriamo l'applicazione fisica del moto rettilineo di un punto. Una vettura (cart) che si muove in modo uniformemente accelerato.

```
#include <stdio.h>
#include <stdlib.h>
#include "population.h"
#include "operators.h"
#include "util.h"
// variabili locali spazio e velocità
Val x,v;
```

```
char x_name[] = "X";
char v_name[] = "V";
Variable X = {&x, x_name};
Variable V = {&v, v_name};
Variable *v_list[2] = {&X, &V};
#define num_vs 2
//  costanti fisiche di forza e massa
#define FORCE_MAG 1
#define CART_MASS 2
// Massimo valore per ogni tentativo
#define MAX_TICKS 2000
// numero di prove di fitness per ogni albero
#define NUM_TRIALS 10
// eguaglianza fuzzy
#define FUZZ 0.1
#define fuzzy_equal(x,y) (((ABS(x-y))<FUZZ) ? 1 : 0)
// funzione per la generazione di costante casuale.
// come proposta da Koza
Val c_rand(void)
{ return ((Val)-1); }
// funzione fitness- simulazione del moto
// del cart (punto materiale)
// su un particolare albero di programmi
// le unità sono state scelte per semplificare
double fitness(Node *n)
{
   // dovrebbe essere eseguito
  //rapidamente, e il fitness totale dovrebbe
   // risultare la media di ogni tentativo
   double trial_fitness = (double)0;
   for (int trial=0;trial<NUM_TRIALS;trial++) {
      // inizialmente la posizione e la velocità sono casuali
      // variano tra -75 e 75 (le unità sono centimetri)
      x = (Val)(int_rand(150) - 75);
      v = (Val)(int_rand(150) - 75);
      // la simulazione si esegue
       // finchè la velocità e la posizione sono
      // approssimativamente zero,
     // o si sfora il tempo di esecuzione
      for (int i=0;i<MAX_TICKS;i++) {
         // controlla se sono verificate le condizioni
         if ((fuzzy_equal(x,(Val)0)) &&
            (fuzzy_equal(v,(Val)0))) {
            fprintf(stderr,"success! x=%f, v=%f\n",x,v);
            break;
         }
         // determina l'accelerazione
       // basata sull'output dell'albero
         Val acc = (FORCE_MAG * n->value()) /
CART_MASS;
         // cambia la corrente velocità
        //  basata sull'accelerazione
         v = v + acc;
         // ottiene una nuova posizione
         x = x + v;
      }
      // determina il fitness –
        // in modo preciso adesso 1/time
        // in caso di successo, oppure -1 se fallito.
        // controlla se è fallito
       if (i==MAX_TICKS)
         trial_fitness += (double)-1;
       else
         // se non è fallito assegna
        // il reale valore del fitness
         trial_fitness += (double)1/(double)(i+1);
   }

   // il fitness totale è la media di ogni tentative
   double avg_fitness = trial_fitness/NUM_TRIALS;
   // restituisce il fitness
   return avg_fitness;
}
main()
{
   // crea un popolazione di misura 20
   Population *P = new
Population(20,b_func_list,num_b_funcs,u_func_list,
num_u_funcs,v_list,num_vs,c_rand);
   // controlla eventuali errori
   if (!P->members()) {
      fprintf(stderr,"errore creando
                                        la popolazione !\n");
      exit(-1);}
   // produce 20 generazioni
   for (int i=0;i<20;i++) {
      // valuta
      P->evaluate(fitness);
      // visualizza il membro migliore
      char *expr = P->print(0);
      printf("generazione  %i migliore (f=%f)->
                        %s\n",i,P->fitness(0),expr);
      delete expr;
      // produce la nuova generazione
      P->spawn(0.3);}
   // ordina la popolazione finale
   P->sort();
   // visualizza il migliore
   char *expr = P->print(0);
   printf("il migliore della popolazione (con fitness
%f): %s\n",
         P->fitness(0),expr);
   delete expr;
}
```

CONCLUSIONI

Vi assicuro che sviluppare un intero progetto e realizzare l'implementazione di un programma genetico non è stato semplice. Tuttavia dopo lo sforzo iniziale tutto dovrebbe procedere automaticamente

Fabio Grimaldi